Information and Software Technology 104 (2018) 72–93

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Diversification and obfuscation techniques for software security: A systematic literature review

Shohreh Hosseinzadeh*,[a], Sampsa Rauti[a], Samuel Laurén[a], Jari-Matti Mäkelä[a], Johannes Holvitie[a], Sami Hyrynsalmi[b], Ville Leppänen[a]

[a] Department of Future Technologies, University of Turku, Vesilinnantie 5, Turku 20500, Finland
[b] Laboratory of Pervasive Computing, Tampere University of Technology, Pohjoisranta 11 A, Pori 28100, Finland

ARTICLE INFO

Keywords:
Diversification
Obfuscation
Software security
Systematic literature review

ABSTRACT

Context: Diversification and obfuscation are promising techniques for securing software and protecting computers from harmful malware. The goal of these techniques is not removing the security holes, but making it difficult for the attacker to exploit security vulnerabilities and perform successful attacks.

Objective: There is an increasing body of research on the use of diversification and obfuscation techniques for improving software security; however, the overall view is scattered and the terminology is unstructured. Therefore, a coherent review gives a clear statement of state-of-the-art, normalizes the ongoing discussion and provides baselines for future research.

Method: In this paper, systematic literature review is used as the method of the study to select the studies that discuss diversification/obfuscation techniques for improving software security. We present the process of data collection, analysis of data, and report the results.

Results: As the result of the systematic search, we collected 357 articles relevant to the topic of our interest, published between the years 1993 and 2017. We studied the collected articles, analyzed the extracted data from them, presented classification of the data, and enlightened the research gaps.

Conclusion: The two techniques have been extensively used for various security purposes and impeding various types of security attacks. There exist many different techniques to obfuscate/diversify programs, each of which targets different parts of the programs and is applied at different phases of software development lifecycle. Moreover, we pinpoint the research gaps in this field, for instance that there are still various execution environments that could benefit from these two techniques, including cloud computing, Internet of Things (IoT), and trusted computing. We also present some potential ideas on applying the techniques on the discussed environments.

1. Introduction

In most organizations, information is a key asset that comes in the form of, for example, financial information, client data, and product design data. Intentional or accidental leakage of any of this information exposes both the business and the customers. Therefore, it is highly significant for any business to have security strategies for protecting the information and services and ensuring the confidentiality, integrity, and availability of the information.

Computer security assures that the system functions under the expected circumstances, and prevents undesired behavior. Many security breaches begin with identifying and exploiting the vulnerabilities in the system. Vulnerabilities are the defects that occur in the process of design and implementation of the software. Defects in design are known as flaws, and the defects in implementation are known as bugs. To ensure the security of software, we need to prevent or mitigate the risk of software vulnerabilities. In other words, we should either eliminate these bugs and flaws, or make it harder to exploit them.

In this paper, we *focus on making exploitation of vulnerabilities harder*, and reducing the possible damage of the attack. To this end, we center our research around two software security techniques, diversification and obfuscation.

Code obfuscation is the process of scrambling the code and making it unintelligible (but still functional), in order to make reverse

* Corresponding author.
E-mail addresses: shohos@utu.fi (S. Hosseinzadeh), sjprau@utu.fi (S. Rauti), smrlau@utu.fi (S. Laurén), jmjmak@utu.fi (J.-M. Mäkelä), jjholv@utu.fi (J. Holvitie), sami.hyrynsalmi@tut.fi (S. Hyrynsalmi), ville.leppanen@utu.fi (V. Leppänen).

https://doi.org/10.1016/j.infsof.2018.07.007
Received 12 May 2017; Received in revised form 2 July 2018; Accepted 7 July 2018
Available online 10 July 2018

engineering more difficult [1]. The transformed code is functionally and semantically equivalent to the original code, but is more complicated and harder to comprehend [33]. With the help of code obfuscation, even if adversaries get access to source code, analysis of the code and finding the vulnerabilities will no longer be a simple task. This requires more time and energy and makes the reverse engineering of the code harder and more costly. Obfuscation does not guarantee that the program is not tampered/reverse engineered, but adds an additional level of defence by increasing the effort and cost for an attacker to learn the underlying functionality of the protected program. Various obfuscation techniques exist that obfuscate different parts of the code at different phases of software development process. For instance, using opaque predicates [75] is a common way of obfuscating the control flow of a program, at source code [109] or binary code level [247], at implementation [109] or compile-time [17].

Software diversification refers to changing the internal interfaces and structure of the software to generate unique diversified versions of it. The users receive unique instances of the software that all function the same, although differently diversified. In other words, diversification breaks the ”monoculturalism” and introduces ”multiculturalism” in the software deployment process.

Malware (malicious software) is any software that intends to run its code on user’s computer to disrupt the computer’s operation or manipulate the system towards the attacker’s desire [2]. To do this, it needs knowledge on how to interact with environment and access the resources. Software diversification alters the internal interfaces of the software and makes it challenging for malware to gain this knowledge. Thus, malware becomes incompatible with the environment and eventually becomes unable to take effective actions to harm the system. It should however be noted that, in order to maintain the access of legitimate applications to resources, we need to propagate the changes to trusted applications, i.e., they will be diversified as well to be compatible with inner layers.

Diversification does not attempt to eliminate the vulnerabilities of a software, but tries to avoid or at least make it toilsome for malware to exploit them and perform a successful attack. In a worst-case scenario, even if the malware succeeds in running its malicious code and attack a computer, this attack can only work on that particular computer. The designed attack model does not work on other computers, since their software are diversified differently with different diversification secrets. To take a large number of computers under control, different attack models should be designed specifically for each software instance, which makes it an expensive and arduous task for the attacker. On that account, diversification is considered as an outstanding approach for securing largely-distributed systems, and mitigating the risk of massive-scale attacks.

It is worthwhile mentioning that the terms obfuscation and diversification, sometimes, have been used interchangeably in the literature. In this paper, we make a clear distinction between these two concepts.

1.1. Method of the study

The method of study we chose in this research is Systematic Literature Review (SLR). A SLR is a means of research that identifies, evaluates and interprets all high quality studies related to a particular research question, or an area of interest [3]. This method of study was originally used in medical sciences [4], but later gained interest in other fields as well. A systematic review can improve a traditional review [4], in a way that the set of studies is not restricted to better-known and frequently-cited publications, and not biased towards the research area/interest of the researcher, as all studies in the field are captured. A systematic review, by classifying and mapping the scattered research studies, identifies research gaps and produce baselines for future research.

We conducted a SLR on studies that deal with the two techniques, obfuscation and diversification, with the aim of securing the code and software. There have been previously some other reviews [5,6,248]. However, they (a) cover a more limited number of studies (14, 69, and 10 papers respectively), (b) consider these two mechanisms from other perspectives than security, (c) focus on one of these two mechanisms, or d) discuss only one particular technique.

The surveys studying the obfuscation related studies include a review on control-flow obfuscation techniques [6], and a review on code obfuscation approaches [5]. These research works cover less than 15 studies and are published, respectively, in 2005 and 2006, which implies that the studies published after that are missing. Larsen et al. [248] authored a survey that reviews the state-of-the-art in automated software diversity with the aim of security and privacy. Another recent literature review on software diversification, surveyed by Baudry et al. [284], investigates diversification from five various perspectives aimed at different goals, including fault tolerance, security, testing, and reusability.

The main factors that differentiate our survey from the existing ones, are: (1) the systematic process for collecting the data, (2) a thorough list of covered studies on both obfuscation and diversification, (3) the focused scope of the study (security), and (4) classification and analysis of the collected studies.

1.2. Structure of the study

The remainder of this paper is structured as follows: Section 2 discusses the aim of our study, and specifies the research questions we have formulated and addressed in this research. Section 3 reports the process of search and selection of the relevant studies, and also the data extraction from these papers. Section 4 presents the results of the data collection and analysis of the results. In Section 5, we present the discussion. Limitations of the study, concluding remarks, and the future work come in Section 6.

2. Aims and research questions

We undertook a SLR of the papers reporting the use of obfuscation and diversification techniques in software security domain. Before starting the search, we determined the research questions, and formed the search strings. Our SLR addresses the following research questions:

- RQ1: What is the *aim* of obfuscation/diversification being used?
- RQ2: What is the *status* of this field of study? (E.g., outputs per annum, types of studies reported, collaboration of academia and industry)
- RQ3 In what *environments* the techniques are used/studied in order to boost the security (i.e., the programming language and execution environment the techniques are used for).
- RQ4: What *mechanisms* have been proposed/studied? (i.e., the obfuscation/diversification method used, (b) target of transformation, (c) level and stage, (d) cost and effectiveness of the approach.

3. Search and selection process

In order to carry out the research review systematically, we need to follow a protocol that defines the search strings and strategy, inclusion and exclusion criteria, and methods to extract data and synthesis the results. In this regard, we based our SLR on the research protocol suggested by Kitchenham et al. [7], and conducted our SLR in seven different phases. These phases are as follow: search and selection process (Phase I), inclusion and exclusion (Phase II to IV), snowballing (Phase V), data extraction (Phase VI), data analysis (VII). Fig. 1 illustrates the different phases in this process. The numbers on the arrows indicate the number of search results and included papers after each phase. In what follows, the details of the protocol developed for our SLR are presented.

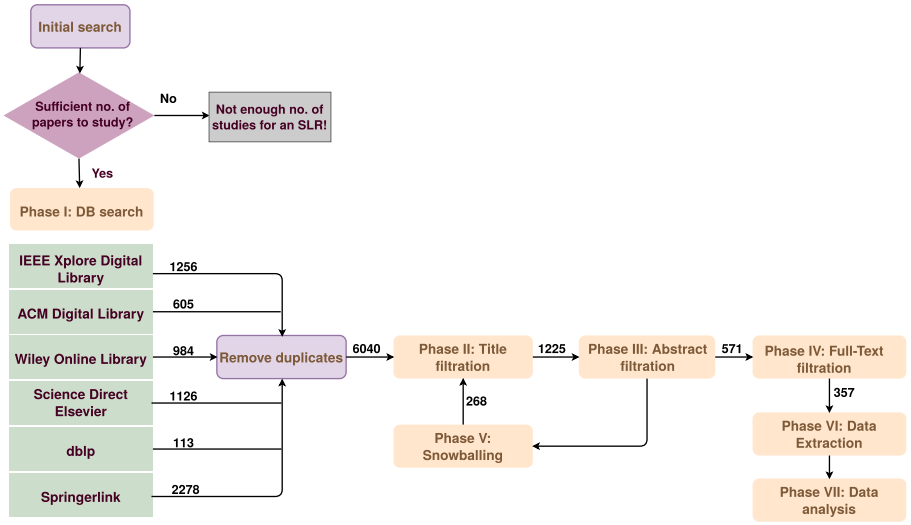

Fig. 1. The systematic search and selection process. On the left are the online databases and on the right are various inclusion and exclusion phases in the study. The number of articles left after each phase are shown on arrows.

3.1. Search

3.1.1. Initial search

Before starting the search process, we conducted an initial search to assure that there are sufficient numbers of articles available in the target field to study. In this stage, we found 48 articles discussing the improvement of software security using diversification/obfuscation techniques, which confirmed that this could be an appropriate topic to conduct a SLR on.

3.1.2. Manual search

For the manual search, Phase I, we selected a set of proper search strings, with which we assumed we would find the majority of the related articles. We also selected six of the largest digital databases, including IEEEXplore Digital Library, ACM Digital Library, Wiley online library, ScienceDirect, dblp, and SpringerLink. We limited our search to titles, abstracts and keywords of the articles to avoid false positive results of the full-text search. In some cases, search query was adapted according to requirements of the search engine. The following search command was used to retrieve studies from the databases:

```
(software OR code OR program) AND (diversification OR obfuscation
OR obfuscate OR obfuscator)
```

We undertook the manual search separately in the databases and combined the results in a large spreadsheet. After removing the duplicates, 6040 articles proceeded to Phase II for inclusion and exclusion (Section 3.2).

3.1.3. Automatic (citation-based) search

To complete the manual search, we performed an automatic search (backward snowballing), in Phase V. Backward snowballing is done by analyzing the reference lists of selected papers to find any missing related paper [7]. Therein, 268 papers were collected, for which we repeated the inclusion/exclusion process (Phase II to IV).

3.2. Selection of the studies

After collecting the papers in Phase I, we should include relevant and drop irrelevant papers. For that, we defined some inclusion/exclusion criteria, based on which we make decision (in Phase II-IV) whether to include/drop a paper. The followings are the inclusion criteria in our study:

- papers that are written in the English language;
- peer-reviewed papers (however, we did not exclude technical reports and books, since there exists some widely cited high quality technical reports in this domain, e.g., [13]);
- papers in the context of software production/development;
- papers related to software security;
- papers related to obfuscation/diversification; and
- obfuscation/diversification in the paper is used/discussed with the aim of improving/enhancing the security in software/code/program.

Considering that obfuscation and diversification techniques have been used in different domains for various purposes, we decided to narrow down our results. To this end, we focused our search on studies that are using obfuscation/diversification with the aim of software security and leave out the papers that were falling in our exclusion criteria:

- studying the possibility/impossibility of obfuscation;
- studying the use of obfuscation/diversification by malware, to hide their malicious code from scanners and malware analyzers;
- studying the techniques at a level other than software (e.g., hardware/network);
- proposing an approach that needs hardware assistance;
- studying obfuscation/diversification from cryptographic point of view;
- using the approaches to protect software watermark, birthmark and intellectual property rights; and
- unavailable studies, that we were not able to access in anyway.

Considering the defined criteria, we followed this process to select the relevant studies:

1. In Phase II, we screened the papers based on their titles. Each paper title was checked by four authors to determine whether it is relevant to our study or not, according to the defined inclusion/exclusion criteria.
2. In Phase III, two of the authors screened the papers based on their abstracts, and included the papers that were compatible with the inclusion criteria and dropped the papers that were not.
3. In Phase IV, the same process was repeated as Phase III, but based on the full text of the papers this time. There were several cases in which the full texts were not available in online databases. We tried

to contact the author(s) or find the text from other sources. If we were not successful finding the text in any way, we dropped the paper.

3.3. Data extraction

Each of the 357 selected papers was read through by two reviewers. The first reviewer extracted the data from the papers using a data extraction form, and the second reviewer checked the correctness of the extracted data. In case of any disagreement, the paper was discussed in a meeting with other authors, till reaching an agreement.

We divided the papers into two main categories, *Constructive* and *Empirical*, and defined different sets of questions to extract data form them. The papers that propose a new (implementable) obfuscation/diversification method, or apply/implement a technique fall into the category of constructive papers. The papers that evaluate/assess/experiment/discuss/review some (existing) obfuscation/diversification techniques fall into the category of empirical papers. There also exist papers that could be considered as both constructive and empirical. This class includes the papers that carry out an empirical study and at the same time conduct a constructive work.

For the category of *constructive papers* we extracted the following data, and presented the classification of the captured data in Section 4.1:

Aim: For what purpose is obfuscation/diversification used and what types of software security problems is solved (e.g., what type of attack is mitigated)?

Level: At what level is obfuscation/diversification applied (e.g., source code, binary level)?

Stage: At what stage of software production is obfuscation/diversification applied (e.g., compile-time, run-time)?

Target: What is the subject of obfuscation/diversification transformation (e.g., control flow)?

Mechanism: What type of obfuscation/diversification method is used/proposed?

Language: What language is the paper targeting?

Execution environment: What environment is the obfuscation/diversification techniques proposed for?

Overhead: What kind of overhead does the proposed obfuscation/diversification technique introduce?

Resiliency: How has the resiliency of the proposed approach been tested, and what results have been achieved?

For the category of *empirical papers*, we extracted the following data, and presented the classification of the captured data in Section 4.2:

Relevance: How is the paper related to obfuscation/diversification?

Outcome: What are the outcomes/findings/results of the study?

4. Results

As mentioned before, based on the method of the study used, we divided the selected studies into three main categories of (a) constructive, (b) empirical, and (c) constructive and empirical. Fig. 2 shows the distribution and the number of papers in each category. As is seen, the highest interest has been on constructive methods and obfuscation studies.

4.1. Constructive studies

By analyzing the data we captured from the data extraction phase, we answered the research questions defined in the beginning of our study.

4.1.1. RQ1: Status of the field of study

After the search and selection step, we extracted data from the 357 included studies. The studies come in six different types, including conference paper, journal article, workshop papers, book section, technical report, and doctoral theses. Also, there were 2 studies in other formats that did not fit into these categories. Table 1 shows different types of studies and the number of studies found in each type. The numbers indicate that the majority of the studies were published in conferences.

We analyzed the author affiliations for the included papers to associate the papers to their originating organizations and countries.

Fig. 3 captures the ten most associated countries for the considered set of studies. *United States* has by far the largest (c. 39, 5%) share, followed by *China* (c. 10,1%). However, as a continent, UK and Europe lead the statistics (c. 40,1%), with research divided mainly among Germany, Belgium, and Italy. The list also includes Japan and India – Asia as a whole contributed to one third (c. 32,2%) of the papers in the study. The research is relatively concentrated to a selected number of regions as the five and ten most affiliated countries count for circa 60,8% and 80,1% of all the affiliations.

Fig. 4 captures the ten most associated organizations for the considered set of studies. From this, we note that *Microsoft Corporation* (inclusive of Microsoft Research) is the only non-academic organization to be prolific in this area. Further, the ten most prolific organizations correspond to almost a third (c. 29, 1%) of the total affiliations for these studies. This is a notable portion from the affiliations, and arguably, indicates that majority of the research is concentrated to a rather small set of organizations. In Belgium, Finland, and New Zealand, the majority of research can be traced to a single organization.

It was of our interest to know the annual growth and decrease rates of the publications in this field of study. This can indicate the changes in interests and the significance of the field of study. An upward trend can be a sign of increasing interest to the field; while, a downward trend could state that the field is reaching a dead end. Fig. 5 illustrates the distribution of the selected studies in the SLR, between the years of 1993 to 2017. There is a relative fluctuation in the whole period, with an overall upward trend in the number of published studies, except for the slight decline in 2017. This implies that while the field has been fairly unpopular research subject, it has recently drawn fair attention among researchers. Between obfuscation and diversification, the former has almost always been a more popular technique – significantly so between 2000–2010, while diversification has grained in popularity since then.

We also examined the articles' publication forum types as a function of their publication years and the distribution is captured in Fig. 6. We note that through the queried year span, the dominant publication forum type is *conference*. However, the type selection gets more varied as we approach the present day, and as a publication forum, the *journal* type is almost on par with the conference in the year 2014. The observed increase in variety could be taken as evidence for the domain getting more mature: existence of more established research in the domain shows as increase in the number of journal articles and book chapters while the discovery of new sub-domains shows as an increasing number of workshop publications.

Fig. 7 displays, for the considered set of studies, the associated organizations' sector as a function of the publication year. Observations made here relate closely to the ones made for Fig. 4: while some publications are affiliated solely to industrial organizations (c. 2, 6% publications in the year 2015 and c. 5, 6% in total for the considered timespan) or to both industrial and academic bodies (c. 13, 2% in the year 2015 and c. 12, 6% in total), majority of the considered studies are made in an academic vacuum. While the distribution is understandable for theoretical research, it raises concerns regarding the applicability and correspondence of the research in this domain.

4.1.2. RQ2: Aim

In the reviewed literature, we identified a set of aims for which obfuscation and diversification were used for securing code and software and defeating known attacks, and hopefully unknown future attacks [238]. In Table 2, we summarize the generic aims that the related

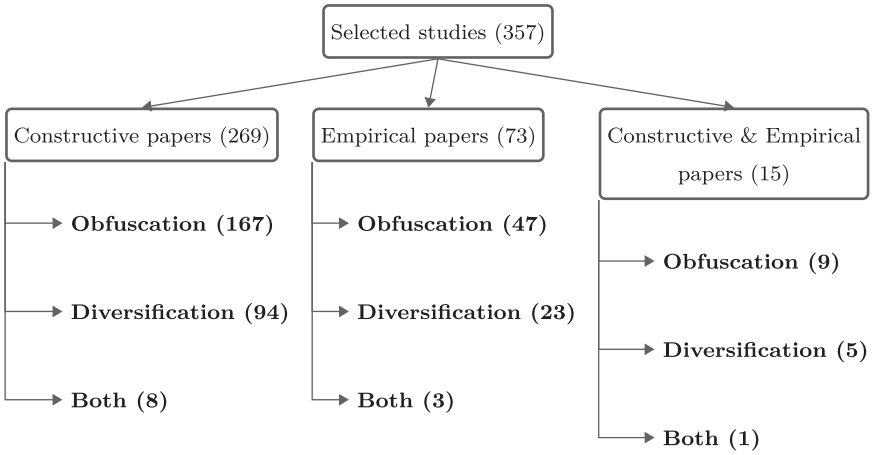

Fig. 2. Distribution of the studies.

Table 1
Types of studies.

Type	Diversification	Obfuscation	Both	Total
Conference paper	68	134	7	209
Journal article	29	51	2	82
Workshop paper	12	17	1	30
Book section	10	8	2	20
Technical report	3	8	0	11
Doctoral Thesis	0	3	0	3
Other	1	1	0	2
	122	223	12	357

studies were following.

In the process of reviewing the selected studies, we identified four broad categories that could encompass most of the presented literature. We acknowledge that these categories are not completely orthogonal, that is, there is some overlap between the different categories and a single piece of research could reasonably be classified as belonging to multiple categories. Still, being aware of the common aims or use cases associated with obfuscation and diversification research can be a valuable resource. With this classification, we try to answer the question what real-world problems are being solved by the use of diversification and obfuscation methods.

a) *Making reverse engineering of the program logic more difficult*: The most commonly stated aim of this research area was simply to make malicious reverse engineering of programs harder [113,165,171,277], i.e., making the act of debugging and disassembling of the software more complex to get the original source code [71,91,123,198,247]. By reducing the readability and understandability [47,110] of the software through these techniques, it becomes more resistant to unauthorized modification, i.e., becomes more tamper-proof [25]. Making understanding programs harder might be a desirable aim in order to protect proprietary algorithms or other intellectual property. Assembly code obfuscation [211], increasing complexity of dynamic analysis [240], preventing control-flow analysis [75], and introducing parallelism in order to obfuscate control-flows [239] are examples of research aiming to make programs harder to understand. Furthermore, obfuscation is an effective approach to counter both static [60,226,268] and dynamic analysis [122,126,240].

b) *Prevent widespread vulnerabilities*: Obfuscation and diversification techniques were also employed for their potential security benefits in preventing widespread vulnerabilities [81,262,268]. Exploits often depend on minute details about program internals. Introducing diversity into deployed applications can make it more challenging to construct exploits that reliably work against multiple targets. Diversification works by introducing variability in the software. Increased diversity makes the number of assumptions an adversary can make about the system smaller. Aside from diversification, obfuscation can also serve as a method of making software more secure. By making it more challenging for an attacker to understand the piece of software, obfuscation helps to increase the costs associated with exploit development. Examples of research specifically targeting security include randomization measures to defeat Return-Oriented Programming (ROP) attacks [216], randomized instruction set emulation [66], metamorphic code generation [230], and diversifying system call interface to defeat code injection attacks [159,233,282].

c) *Preventing unauthorized modification of software*: Research on tamper-resistance tries to find ways for making it more challenging for an adversary to produce derived version of programs [26,107,127].

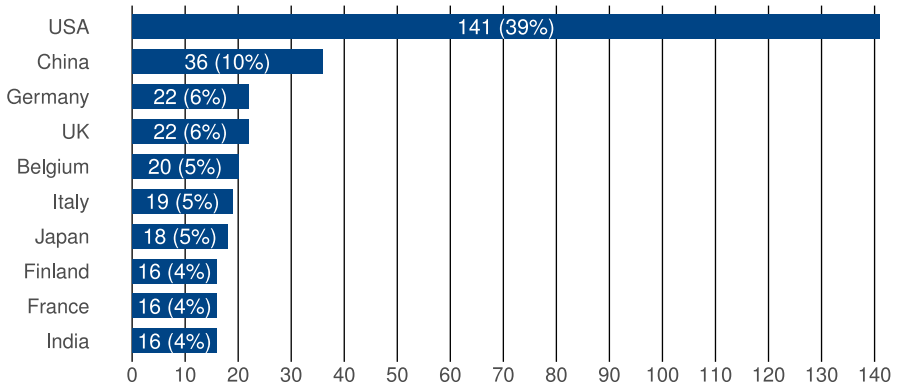

Fig. 3. Prolific countries: ten most associated countries in the considered studies (total number of country level affiliations $N = 420$).

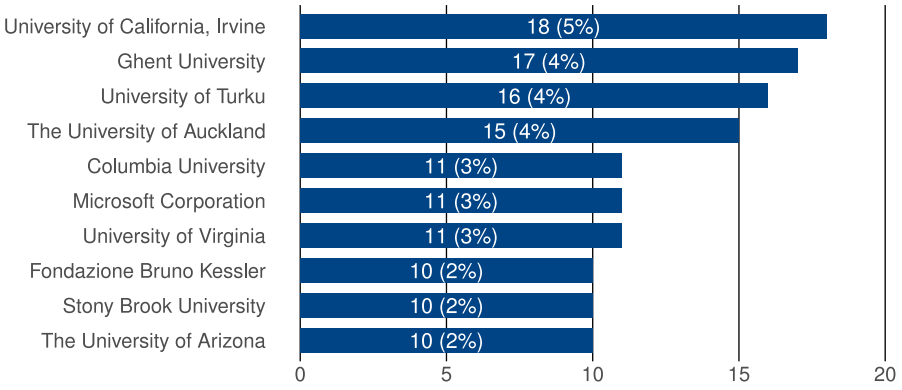

Fig. 4. Prolific organizations: ten most associated organizations for the considered set of studies (total number of organization level affiliations $N = 544$).

This might be desirable in order to preserve the intended operation of a program in an uncontrolled environment. For example, applications employing some form of digital rights management or computer games trying to prevent players from cheating might employ such techniques in order to make it harder to circumvent the protection mechanisms [30,259]. Techniques aiming for tamper-resistance often utilize methods for making understanding the program more difficult but they can also include methods for verifying program authenticity. Tamper-resistance was explicitly mentioned as one of the aims in the context of obfuscating Java bytecode [30], run-time randomization in order to slow down the adversary's locate-alter-test cycle [103] and obfuscation of sequential program control-flow [24]. Control flow obfuscation conceals the real control flows of the program and generates a fake control flow [145,175]. This makes it difficult for an analyzer to comprehend the logic of the program [245], also prevents spying and manipulating the control flow [75].

d) *Hiding data*: Aside from making programs more complex to analyze, obfuscation was also utilized for hiding static non-executable data within programs [99,231,281]. Hiding cryptographic keys and protecting intellectual property are few examples of scenarios were such measures are considered. Such techniques have been used to hide static integers [138,191] and obfuscate arrays by splitting them [97].

The results signify that the two techniques are used to mitigate the risk of a wide range of attacks, and in best case scenario hamper them. Table 3 presents the top attacks that were impeded with the help of obfuscation and diversification, such as code injection attacks [55,105,108,197], ROP attacks [195,215,260,263], buffer over-flow attacks [35,57,268], and Just-in-Time (JIT) spraying attacks [186,208,263]. From Table 3 we can deduce that not all the studies (209 papers) were explicitly discussing particular attacks that they aim to impede.

4.1.3. RQ3: Environment

For classifying the environments, two subcategories were chosen: a) *language* of the program being obfuscated/diversified and b) *execution environment*.

a) *Language*: The reviewed literature used a diverse set of over 20 different programming languages. Circa 36,8% of the languages were the topic of only one research and two thirds (63,1%) were mentioned at most thrice. Most research discussed one (c. 63,4%) or two (c. 10,6%) specific languages, with two systems programming (C/C++) or high level languages (Java & JavaScript) representing the vast majority of such pairs. A quarter (25,0%) of the research did not specify a single language or generalized the presented work for a class of languages. Only a minority of research [135,158,163,167,191,232,340] mentioned multiple languages or language classes.

A more descriptive view of the kinds of the languages was achieved by further classifying the research into four language categories representing hardware oriented, high level, scripting, and domain specific languages. The distribution of languages into these languages is as follows:

- Systems programming (N = 158): C (52), Assembly (29), C++ (21), Cobol (1)
- Managed (N = 81): Java (54), C# (3), Haskell (2), J# (1), Lisp (1), OCaml (1), VB (1)
- Scripting (N = 19): JavaScript (11), Python (3), Perl (2), PHP (1)
- Domain specific, DSL (N = 7): SQL (5), HTML (1)

The systems programming languages are compiled to native hardware without a run-time virtual machine and provide direct access to memory. Due to this low level direct hardware access, these languages benefit from obfuscation and diversification to protect this access. Some

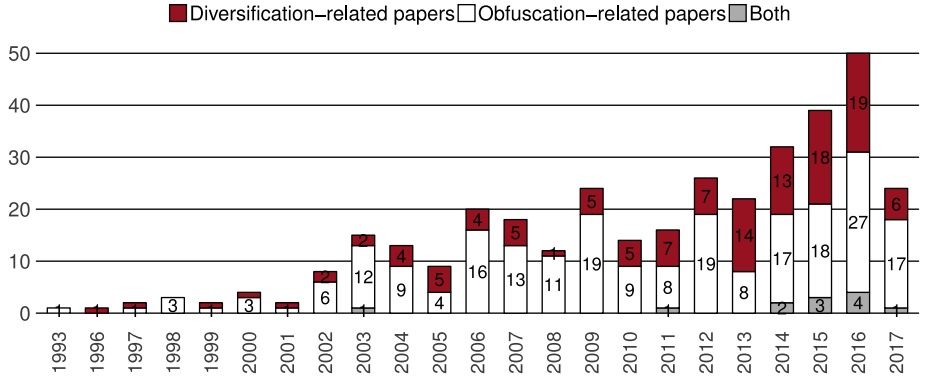

Fig. 5. Number of papers published yearly on the topic of security and privacy through obfuscation/diversification.

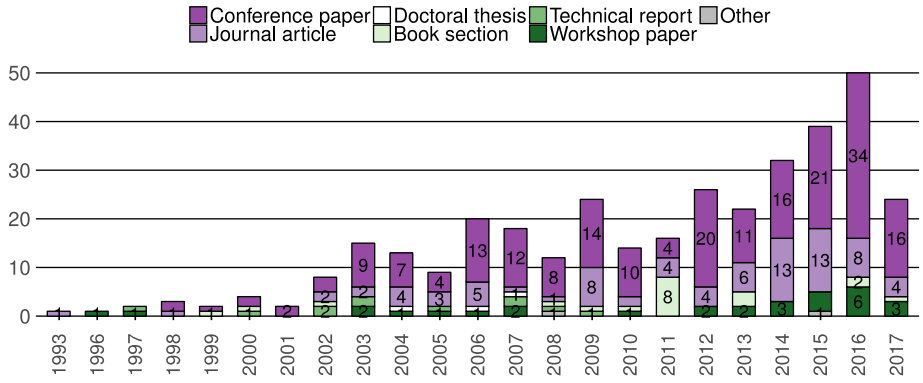

Fig. 6. Publication forum types for the considered set of studies as a function of the publication year.

examples of the applications of these languages in the research include operating systems and drivers, low-level libraries, server software, high performance computing, and embedded software. The managed languages typically require a virtual machine to provide a safer programming model for application programming. The most common problem for these languages is that the code is relatively easy to reverse engineer. The Java virtual machine is the most common platform in the selected research studies, but others, such as the Microsoft's Common Language Runtime (CLR), were also covered. A typical application of this class is mobile code, that is, code expected to run in an unknown environment. Finally, the scripting languages introduce new levels of insecurity since manipulating their code is even simpler. The DSLs have other issues, for example injection attacks or the need to protect intellectual property.

The following three figures present the language trends in the reviewed papers. First, Fig. 8 shows the popularity of various language categories based on our classification. The majority of the research has focused on systems programming, managed languages come as the second most popular category. Script languages are a bit more researched than DSLs.

Fig. 9 shows the overall distribution of language popularity in selected studies. A raw binary code (of native or virtual machine bytecode instructions) is the most popular "language" in this field of research. This is natural as most software is compiled to binary form for distribution. It represent the lowest level language and often requires disassembly to reconstruct the program structure for analysis. We distinguish assembly language as a separate form with its structured form intact for further analysis. Assembly is commonly used when obfuscation/diversification is used as a language agnostic compiler pass. The C and Java languages are other popular choices, followed by C++ and JavaScript.

Fig. 10 shows the trend over time for the five most used languages. The other languages are presented as the sixth group, as a reference. Like in the other figures, the research seems to be a bit more active in the 2000s and even more active in 2010s. Each of the top five languages appears to be almost equally represented each year.

b) *Execution Environment*: The environments in the reviewed literature can be classified in various ways as there are many interesting areas of focus. We have focused on two approaches in our review. First, the target environment of deployment (Table 4) plays a significant role when analyzing the applicability of a security mechanism. The majority of reviewed approaches are general enough to work in a multitude of environments. The most significant group of special environments were distributed and agent based systems with mobile code. As the code executes in a possibly remote, uncontrolled system, the need for protection is obvious - especially since the mobile agents often rely on bytecode that is relatively easy to reverse engineer. Virtualization and cloud computing can introduce similar kinds of problems if the host is owned by a third party, but virtualization is also used as a protection mechanism. Web services and servers offer an attack surface via the service layers, and mobile and desktop users are threatened by unreliable software. We distinguish between generic servers and cloud by denoting XaaS platforms for hosting third party services as the cloud. Embedded environment might use obfuscation or diversification for example to avoid the computational cost of encryption. Furthermore, most mobile devices are embedded platforms, but not all embedded platforms are mobile.

The second way to classify the reviewed literature is by the run-time environment (Table 4). This classification focuses on the abstraction level on the deployed software stack, with native code on the bottom and the virtual machine managed code on top, if both run-times are being used. Over a half of the research targets a native code environment. The more specific mechanisms are further discussed in the level

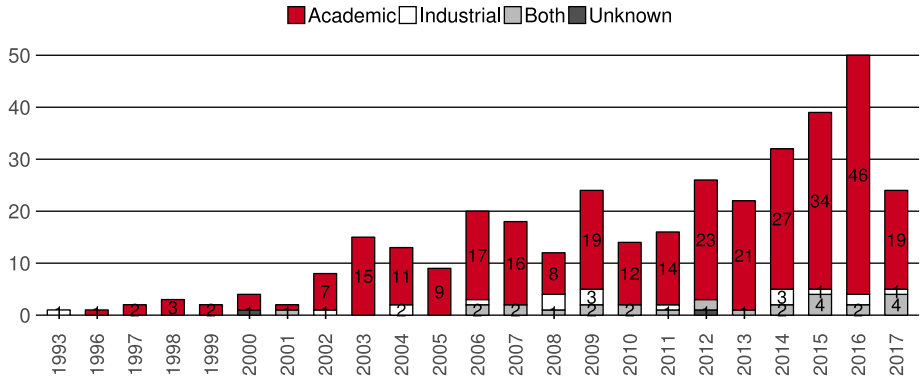

Fig. 7. Associated organizations' sector for the considered set of studies as a function of the publication year.

Table 2
Aims followed by using obfuscation and diversification techniques.

Aim	Via diversification (no. of papers)	Via obfuscation (no. of papers)
Making reverse engineering difficult	7	78
Generating diverse and unique versions of SW	34	3
Making the program hard to comprehend/read	1	31
Concealing a fragment of code and hiding some data inside the code	2	24
Preventing tampering of program code and illegal modification of software	4	22
Hiding the control flow of the program	1	24
Making static analysis difficult	1	20
Making dynamic analysis difficult	2	12
Mitigating the risk of malware	12	7
Protecting mobile agents against malicious host	0	6
Preventing large-scale attacks	10	2
Detecting anomalies/intrusions	4	0
No suitable aim discussed	50	

Table 3
Attacks mitigated by obfuscation and diversification techniques.

Attack mitigated	via diversification (no. of papers)	via obfuscation (no. of papers)
ROP attacks	24	1
Code injection attacks	15	2
Buffer overflow attack	6	2
JIT spraying attacks	2	2
Side channel attack	3	4
Attacks to web applications, e.g., cross-site scripting (XSS), SQL injection	4	1
Code reuse attacks	12	2
Browser-based attacks	2	3
Insider attacks	1	2
Protecting the software against piracy	0	6
Slicing attacks (a form of reverse engineering)	0	2
No attack mentioned	209	

(Section 4.1.4b) and stage (Section 4.1.4c) sections. Around fifth of the research focuses on managed environments such as Java virtual machines. Few papers target both environments, e.g., in the case of JIT compilers. Almost a third of the research claims to operate in all kinds of environments as a general purpose security mechanism.

4.1.4. RQ4: Mechanism

a) *Method*: In order to make diversified program instances, various transformation mechanisms are proposed in the literature. Each of these mechanisms are applied at different stages and levels of software development life-cycle (discussed in Section 4.1.4.b and Section 4.1.4.c). In this section, we classify the transformation techniques, based on the target of transformation. In other words, "what" is transformed and "how" the transformation is applied. Fig. 11 illustrates these techniques as a tree. On first level of the tree come the targets of transformations and on the lower levels the transformation techniques to obfuscate these targets. We base our classification on the taxonomy presented by Collberg et al. [13], which introduces *control obfuscation, data obfuscation, layout obfuscation*, and *preventive obfuscation* as different transformation targets. In the following we discuss each category.

- *Control flow obfuscation* aims at altering/obscuring the flow of a program to make it difficult for an attacker to successfully analyze and understand the code [1]. There exists a large body of research on control flow obfuscation techniques [259,261,266,334]. The most common technique to disturb the control flow is bogus insertion [11,12,22,31,63,73,104,268,362]. This technique works as inserting gray/dead/dummy code [351] that is never executed, fakes the control transfer [100], and/or introduces confusion for the analyzing tools [16,24,45,51,98,109,129,145,179,187] to attain the actual flow. Adding dummy blocks [122,160,169], dead statements [170], redundant operands [113], dummy instructions to camouflage the original instructions [38,242], new segments [247], dummy classes [84], dummy sequence using dead registers [47], and junk byte insertion to instruction stream [34,169], all fall into this class of transformation. Inserting additional NOP instructions [215,226,283] is another type of bogus insertion. NOPs are instructions that perform no operations but make it harder to predict where the pieces of code are placed in memory.
  Another widely used technique for disturbing the program's control flow is using opaque predicates [16,34,73,75,115,126,145,169,179,188,209,291,313,355]. These expressions are known to the obfuscator in advance, but not to the deobfuscator/attacker. A simple example of opaque expression is a Boolean expression that is

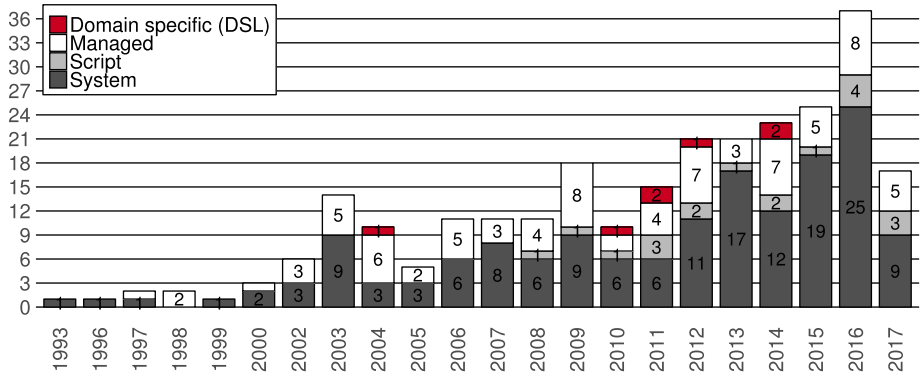

Fig. 8. Popularity of languages in the selected publications over time, grouped in language categories.

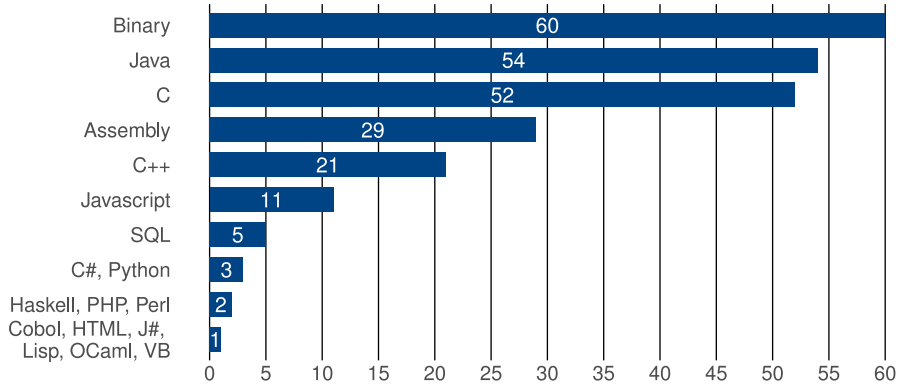

Fig. 9. List of languages in selected research, ordered by their popularity over time.

always evaluated as ”true” or as ”false”, yet needs to be evaluated at execution time. This hardens the task of analyzing the control flow and enhances the cost of comprehending the program [16,75].

Transformation can be applied to loops [268] by loop unrolling [166,201,272], loop intersection [73,182], extending [16,104,113] and eliminating [109], and changing the loop conditions [330]. Transformation can also be applied at instruction level to camouflage the original flow of the application [271] through instruction reordering [103,114,166,245,268], instruction hiding [226], and instruction replacement with dummy/fake instructions [38,76,175,291,346], or instructions that raise a trap [47,103,186,247]. Self modification mechanisms [38,62,165,182,202] alter/replace instructions at run-time which could be used to introduce an additional layer of complexity while obfuscating the code [161].

Modifying the control of a program not only makes it difficult to analyze the actual program's flow, but also results in diverse binaries/executable. This can be achieved through reordering the instructions [103,114,166,245,268] and blocks [27,103,135,202,239,346], while the semantics and dependency relations are preserved. Code transformation [52,63,162,202,211,214,230] is another way of producing dissimilar binaries. As an example, by randomizing the software in a sensor network, the nodes receive diversified versions of the software [149].

Other forms of control flow obfuscation are polymorphism [44,84], branching functions [34,47,123,157,179,209,240], and transforming/faking/spoofing jump tables [34,242]. Inlining method [41,103] replaces the function call with the function body, so the function is eliminated and the primary structures are not disclosed. Cloning method [229,231] creates different versions of the function and tries to conceal the information about the function calls.

- *Data obfuscation* aims at obscuring data and concealing data structure of a program [207]. In the surveyed studies, various approaches have been used to this aim [259,279,288]. *First* is array obfuscation [29] that targets the structure (and the nature) of an array, trying to make it confusing to the reader. This can be done through splitting an array into smaller sub-arrays [97,112,130,171], or merging multiple arrays and making one larger array [130,171]. Other ways of array obfuscation are array folding [85,112,130,171], that increases the dimensions of an array, and conversely, array flattening [48,85,112,130,171], that decreases the dimensions of an array. *Second* is variable transformation to obscure/obfuscates variables [29,41,67,110,116,238,256]. Variables can be encoded [104], substituted with a piece of code [11], split into multiple variables [94,104,113], and vice versa, multiple variables can be merged together. *Third* is a more complex obfuscation technique, class transformation, which confuses the reader to comprehend the structure of a class [72]. This transformation includes class splitting into smaller sub-classes [36,41,128,177], merging/coalescing multiple

Table 4
Environments for the proposed obfuscation and diversification mechanisms.

Target environment context	Diversification	Obfuscation	Both
Cloud	5	3	2
Desktop	1	3	0
Distributed/agent based	18	9	2
Embedded	6	2	3
Mobile	13	5	1
Server/mainframe	4	12	0
Virtualization	7	7	1
Web	10	8	0
Runtime environment			
Any	54	21	6
Managed code	46	8	1
Native code	72	68	2
Both native & managed	4	1	0

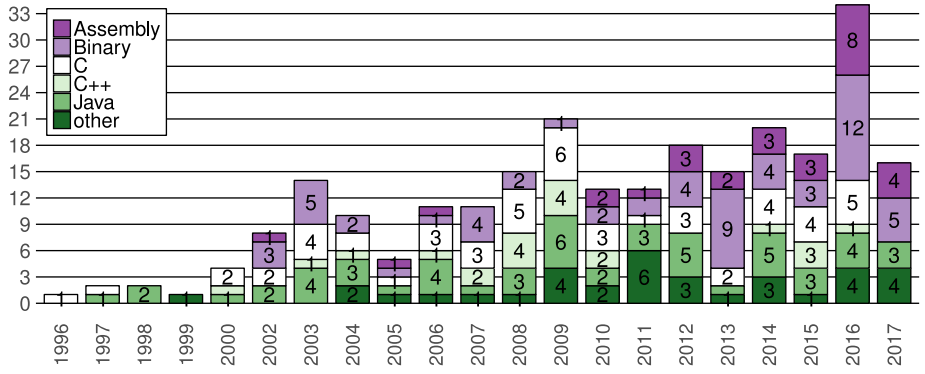

Fig. 10. Popularity of top five most used languages in the selected publications over time, the other languages are merged to the sixth group.

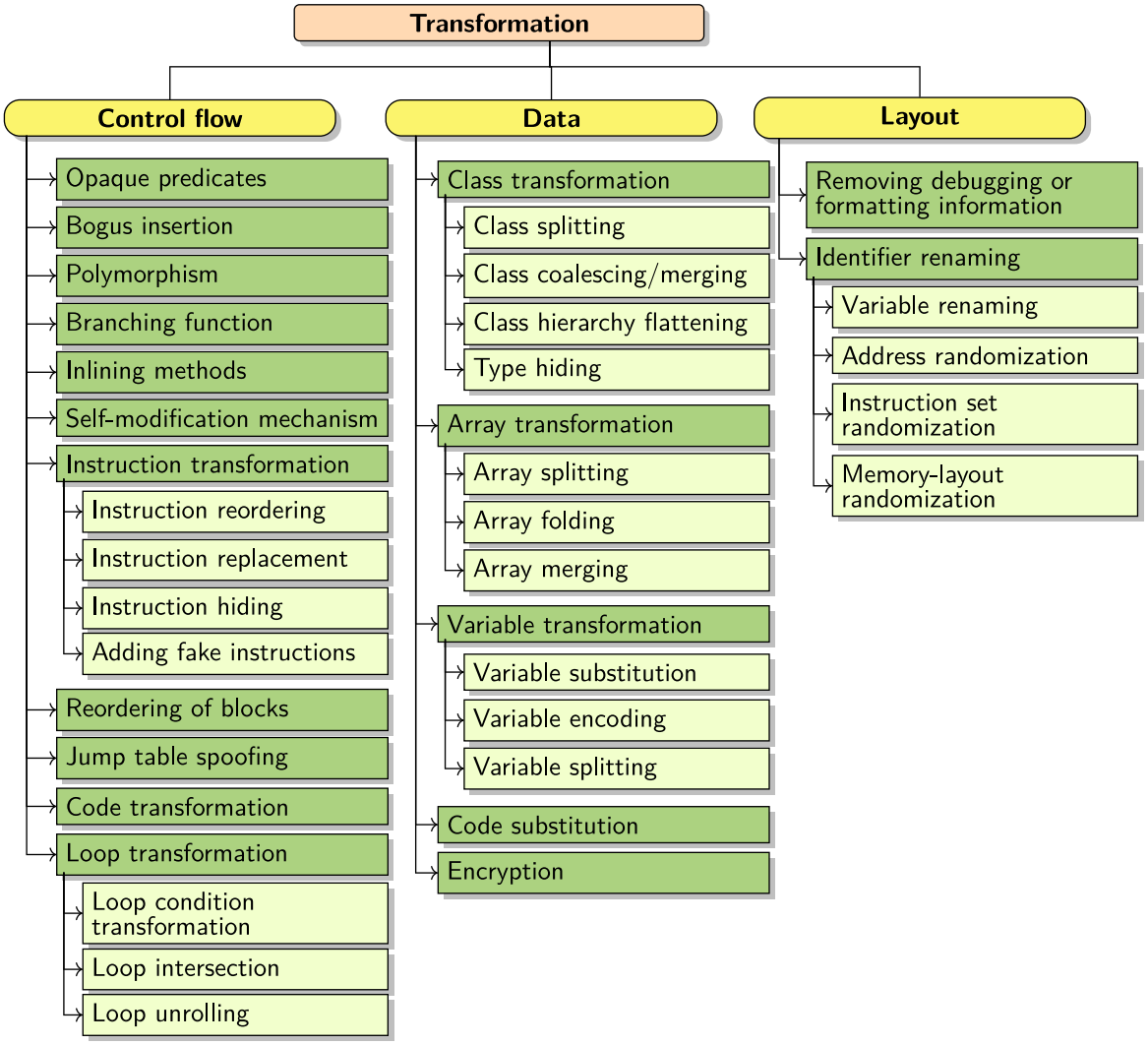

Fig. 11. Transformation mechanisms.

classes together [36,41,148,177,223], class hierarchy flattening [84,128,223] which removes type hierarchy from programs, and type hiding [36,72,177]. There exist other classes of techniques to obfuscate the data structure of the program, such as code substitution [145], and encryption [53,67,86,110,128,147,213,235].

- *Layout obfuscation* is a class of obfuscation techniques that targets the program's layout structure [13,336] through renaming the identifiers [45,51,98,101,110,117,125,163,187,212,213,233,320] and removing the comments, information about debugging, and source code formatting [113,170,201,223]. By reducing the amount of information for the human reader, the reverse engineering becomes harder. Layout transformations are considered as one-way approaches, as when the information is gone there is no way to recover the original formatting. Instruction Set Randomization (ISR) [55,66,105,140,154,158,167,186,192], Address Randomization [35,39,46,57,105,106,192,193,215,283], and Layout Randomization [41,52,88,113,146,149,160,178,193], Address Space Layout Randomization (ASLR) [263,265,308,337] can also be seen as identifier renaming techniques.

b) *Level*: We identified several phases in the software development, deployment, and execution as levels of obfuscation. In the reviewed research (Table 5), most techniques apply to development time (n = 282), runtime (n = 95), or both (n= 58). The development time techniques mostly apply to human readable source code (high level language & assembly), but obfuscation and diversification tools manipulating the generated binary formats (bytecode, native code, intermediate representation) are equally common. The application program itself provides the main platform for applying various mechanisms. At runtime, the techniques either target the source code (scripting languages), intermediate formats (e.g. JIT compilation), or the execution environments. Modified runtime systems are process level techniques for both managed (e.g. CLR & Java virtual machine) and native code,

Table 5
Level of obfuscation and diversification at development time / runtime.

Level	Development	Runtime
Application design	11	–
Assembly source code	12	–
Bytecode	40	–
Executable	76	–
High level language source code	104	7
Intermediate representation format	39	5
Managed code	–	3
Native code	–	43
Hardware	–	3
Operating system	–	18
Virtualization	–	16
Total no. of papers (impl & runtime effects)	58	
Total no. of papers	282	95

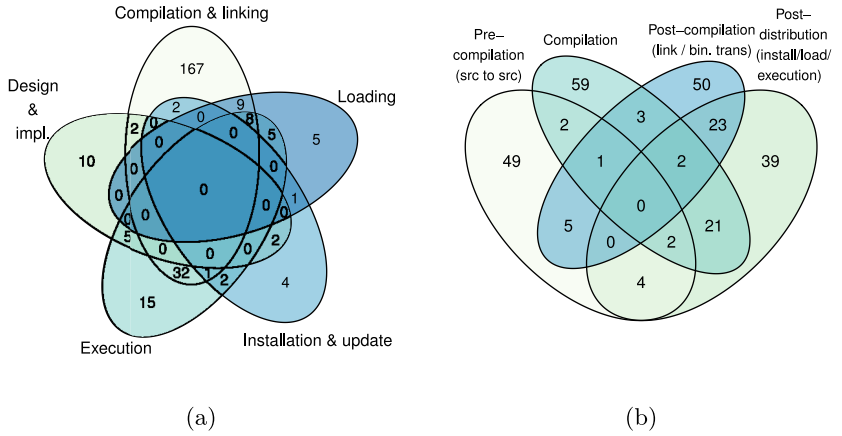

Fig. 12. a) Various stages in SW life-cycle that obfuscation/diversification are applied on, b) Dissection of various compile-time stages in conjunction with all post-distribution phases.

but operating systems, operating system / machine virtualization, and hardware level modifications are also presented.

In terms of obfuscation and diversification techniques, operating with *source code* that is not yet compiled is relatively effortless. Many of the reviewed techniques work purely on the lexical and syntactic levels and the parsing technology is mature, ranging from simple pre-processors to frameworks with compiler-like abilities. It is also possible to manipulate many high-level structures (classes, data structures) that are not available in machine code form [36]. In interpreted languages (e.g. JavaScript), the source code obfuscation is the only option [110], which also explains why some of the source code obfuscations are deferred to run-time. Collberg et al. have extensively described techniques available for source code obfuscation in [13,16]. Some of the mechanisms extend the range of obfuscations to semantically richer forms, the intermediate formats available during the compilation. Abstract syntax trees [133,207] are used by syntax oriented techniques while semantically richer intermediate formats provide access to e.g. control flow analysis. These mechanisms are provided for instance as compiler plugins.

The motive to obfuscate the source code is usually preventing the adversary from easily understanding and altering the code even if he or she has managed to reverse engineer it. Source code obfuscation might not ultimately prevent a dedicated attacker from understanding software, but it will significantly raise the bar of complexity and decrease the probability of a successful attack [49]. Source code obfuscation is often used for intellectual property protection [104,113]. Worth noting is that source code obfuscation is usually also reflected to the bytecode or binary code after compilation.

In managed environments, *bytecode* techniques have received lots of attention. For example, in Java, it is not that hard to reverse the compiled bytecode back to source code. This reverse-engineering can be performed via automatic tools [50,51]. Naturally, this poses problems for the confidentiality of source code and has elicited lots of research on bytecode obfuscation. Several approaches such as [30,45,177,182,223] have been proposed to prevent adversaries from understanding, reverse-engineering or cracking the bytecode. One major advantage of bytecode obfuscation (along with other binary code obfuscation techniques discussed next) is that source code is not needed in the process. This is quite often the case with closed source, third party software.

Reverse engineering and the manipulation of security measures are also issues with native code executables, but the native instruction sets are inherently harder to analyze due to more complex instruction sets and the lower level of abstraction. A large set of obfuscation and diversification techniques are applied to symbolic assembly code (with relocation information etc. intact) or disassembled final binaries [259,276]. This is often done in order to make reverse engineering considerably harder [171,247] or to prevent disassembling the program from binaries [175,240]. Low level obfuscation usually involves using control flow obfuscation transformations changing the sequence of instructions [123,245]. In general, increasing the entropy of the low level code also makes it harder for a piece of malware to modify the code or inject its own malicious payload [11,233]. One technique related with low level obfuscation is ISR [42,66]. An execution environment unique to the running process is created so that the attacker does not know the ”language” used and therefore, cannot ”talk” to the machine. A new instruction set is created for each process executing within a system.

c) *Stage*: Although modern software development is iterative, we observe the software life cycle as a linear sequence of stages: (a) development, (b) distribution and deployment, and (c) execution. This model captures the fact that each stage is characterized by a different set of obfuscation and diversification techniques and tools. The development stage is further split into design, implementation, compilation and linking phases. When analyzing the types of tools used to manipulate the application’s code, the compilation can be further refined into pre- (e.g. source to source transformations and code generators) and post-compilation (e.g. link-time code transformation) phases. The software deployment includes installation and updates [248]. Application loading occurs in conjunction with execution and thus is included in this stage. The surveyed studies discussed and applied obfuscation and diversification techniques during all these stages.

The Venn diagram in Fig. 12a illustrates all the observed stages and their overlap in five main groups, from design to application execution. These groups reflect the different stakeholders and roles in the software’s life cycle. We identified 16 different types of use of stages, with 201, 60, and 9 studies operating on one to three stages, respectively. None of the studies suggested taking part in four or more groups of stages. A majority of research involves compilation and linking. Execution time techniques form another large group. A small number of research is associated with either of these approaches and some other stage (n = 29) or is applied outside these stages (n = 22).

In Fig. 12a, the first group contains design and implementation phases. Mechanisms applied at this stage are involved in software development effort. Data obfuscation [118,137,162], control flow obfuscation [109,169] and, in general, source code obfuscation [63,99,118,188,225] are the most common approaches that target the code at *implementation level*.

The mechanisms in the next group, compilation and linking, can be applied to the deliverables of iterations for in-house software or to premade software, available either as source code, in intermediate forms, or as executable binaries that can be analyzed or reverse-engineered. This group is further dissected in Fig. 12b as the majority of reviewed

literature forms a cluster in this stage. The third stage, installation and update, includes the task of local deployment of software and updates. The next stage, loading, covers the process of loading the executable to memory (e.g. from a network stream or disk) and dynamic linking. Finally, execution stage includes all sorts of mechanisms that activate during application's execution. Code obfuscation and software diversification can also be applied at *execution time*. Dynamic software mutation [79] is a repeated transformation of the program during its execution. It makes a region of memory occupied by various code sequences during execution. Identifier renaming [163], ASLR [57,265], camouflaging the instructions by overwriting them with fake instructions [76], and randomizing the location of critical data elements in memory [140] are other examples of execution-time diversification.

Fig. 12b focuses on the mechanisms applied on various stages of the compilation (pre-, post-, and during compilation). In the figure, all the remaining stages after the compilation techniques have been combined as a single post-distribution stage. The reviewed research is distributed quite evenly between the different compilation stages. Second large class of mechanisms is to use compilation or post-compilation in conjunction with the execution time techniques.

Diversification at the *compile-time* makes the process fairly automatic by eliminating the need to change the program's source code [248]. M. Franz [150] has proposed a practical approach for generating diverse software at compiler-level. This approach is based on an app-store that contains a multi-compiler, which works as a diversifying engine that generates unique binaries with identical functionality. In [207] they have developed a compiler plugin to generate diverse operating system kernels, through memory layout randomization. As we mentioned before, in the literature, there are several works that study the pre-compile time and post-compile diversification. Control flow transformation in the source code [43] is an example for the former, and class transformation in Java bytecode [177] an example for the latter.

d) *Cost*: Despite of the security obfuscation and diversification bring, they introduce cost and overhead to the system, like any other security measure. In fact, the higher level of obfuscation/diversification, the more penalty is forced to the system. Therefore, based on the need of the system, it is decided how much the program needs to be obfuscated/diversified. In the studied works overhead mainly was reported as a) increase in the program size [50,84] (e.g., number of instructions [34], memory size, code size [240,256,264], binary patch size [140], byte code size), b) increase in program performance [261,263,266,285,290] (e.g., compile time [175], process time, execution time [260,268], CPU overhead [119], higher memory usage [119,273], and c) latency and throughput [210] (in load time or run time). It is worth mentioning that among the diversification mechanisms, some introduce more cost and some less. For instance, changing variable names, function names, and system call numbers often introduces no additional costs.

e) *Effectiveness*: In the studied works, the effectiveness of the proposed approaches were mainly measured through the following metrics:
- *Potency* determines to what degree a human reader is confused, as a result of the applied security measure [13]. Measuring the potency can be done by comparing the obfuscated/diversified version of the software with the original version and presenting the similarity rate [257,290]. In [131] clone detection is used to analyze the similarity of the obfuscated code with the original one, and the code dissimilarity is the metric for representing the potency of the approach. Another way to measure the potency of an obfuscation mechanism is to evaluate how much harder it has become for a human reader to comprehend the obfuscated code, comparing to the original code. For instance, the obfuscation mechanism in [27] has been tested empirically with a group of students, programmers and crackers and illustrated that only a few crackers were able to deobfuscate the obfuscated code. In [272] the effectiveness of the proposed approach is measured by static and dynamic analysis of the obfuscated code.
- *Resiliency* determines how well the obfuscated/diversified program resists automatic decompilers/disassemblers/deobfuscators [13]. Analyzing the reverse engineering effort demonstrates how the proposed technique is effective against disassembly tools (e.g., through presenting confusion factor, and disassembly errors). For instance, in [211,240] the strength of the obfuscation mechanism has been evaluated against IDA PRO automated deobfuscators [8], and demonstrate that obfuscated code increases the effort for an attacker, by making it harder to reconstruct the original code. Similarly, Linn et al. [34] have used three state-of-the-art disassembly tools, and demonstrated the effectiveness of their approach through confusion factors, disassembly errors, and incorrectly disassembled code, that they gained by disassembling the obfuscated code.
- *Attack Resistance* determines how much harder it has become to break the obfuscated code. It can be done by running the obfuscated/diversified software against different attacks and analyzing the outcome [260,263,268,285]. As an example, the obfuscated kernel in [207] is tested against four kernel rootkits, and it is shown that they all were disabled. RandSys prototype [105] implemented for Linux and Windows has been tested against two zero-day exploits (code-spraying attacks), and 60 existing code injection attacks. It was shown that the approach is successful in thwarting them. In [268] they run the program against various types of attacks (e.g., code injection, memory corruption, code reuse, tampering and reverse engineering attacks) and measure the resistance.

4.2. Empirical studies

As mentioned before, in the set of studies collected, 68 of them were studying the obfuscation/diversification techniques empirically. These empirical studies come in the form of discussion, experiment, evaluation, comparison, optimization, survey, and presenting a classification. The following categories illustrate how these studies were related to obfuscation and diversification:

- survey of related works on obfuscation and diversification as software protection techniques [5,6,37,64,180,248,284]; Baudry and Monperrus [284] survey the related works on design and data diversity which consider fault tolerance and cybersecurity. They also study randomization at various system levels.
- overview/classification of existing obfuscation/diversification techniques [59,78,132,324];
- studying the obfuscating transformations that are (more) resilient to slicing attacks [92,96,329];
- comparing different obfuscation mechanisms [87,95,190];
- discussion on a particular obfuscation mechanism [19,78,132]; In [132], obfuscation is being discussed as a way to make understanding the software more difficult. In [19], identifier renaming is discussed as an obfuscation mechanism to protect Java applications. By making the classes harder to decode, the act of unauthorized decompilation becomes difficult. In [78], the authors overview the existing obfuscators and obfuscation mechanisms, and also illustrate the possibility of achieving binary code obfuscation through source code transformation.
- studying/evaluating the effectiveness of an obfuscation/diversification approach (e.g., identifier renaming and opaque predicates) against human attackers [54,64,74,93,176,183,246,266,269,278,289,295]; In [68] the authors qualitatively measure the capabilities

and performance of two commercial obfuscators for three different sorting algorithms. In [93], the effectiveness of decompilers and obfuscators are quantified through a set of metrics. It is done by comparing the original Java source code with the decompiled and obfuscated code respectively. The metrics will then measure whether the decompiler produces valid source code, and whether the obfuscator produces garbled code. [176] measures the effects of different obfuscation techniques on Java code in terms of complexity. In [278] several different metrics are suggested for measuring the incomprehensibility of the obfuscated code.

- optimizing and reducing the overhead of software diversity [83,202,237,255];
- experimenting and evaluating the potency of an obfuscation technique; The strength and incomprehensibility of the obfuscated programs can be evaluated by measuring the performance of human analyzers in analyzing the obfuscated code (to what degree a human reader is confused) [111,121,145,228,234,354].
- studying the effectiveness of software diversity [32,65,136,237,274,287,292,360]; For instance, to evaluate the effect of diversity, several different computer attacks are tested against the diversified programs [32]. In [274], automatic software diversity is discussed as a means for securing the software. The authors investigate the types of exploitation it can mitigate, the different levels of software life-cycle the diversification can be applied at, and the possible targets of diversification.

5. Discussion

The idea of protecting software through generated diversity and obfuscated code originated in early1990′s and gained more attention in the past decade. The rationale behind these techniques is to increase the cost and effort for a successful attack. This study, by surveying the literature about the use of these two techniques for securing software, elucidates several points.

First, these methods have been used in various ways with different aims, such as protecting software from malicious reverse engineering and tampering, hiding some data and protecting watermark information, preventing the wide spread of vulnerabilities and infections, mitigating the risk of massive-scale attacks, and impeding targeted attacks. In a previous study, we have studied the aims and environments that these two techniques have been applied to [324].

Second, the field has grown in many directions, and new areas have emerged. Moving Target Defence (MTD) [172] is an example of newly born defence mechanisms. MTD randomizes the system components, and presents a continuously changing attack surface, which shortens the time frame available for attacker.

Third, studying all the related works sheds light on the research gaps, and the potential research directions. We discuss these research gaps in Section 5.2.

Fourth, There are also some challenges associated with practical diversification and obfuscation that require further study. In Section 5.1 we discuss some of these challenges.

5.1. Challenges

One of the challenge of applying diversification is the fact that it has to be propagated to every parts of the system that makes use of the diversified interface. For example, diversified system call numbers in the operating system kernel have to be propagated to libraries that employ system calls. It is not straightforward to automatically find all the dependencies.

Software updates are also a challenge for diversified systems, because each update has to be individually diversified to be compatible with the uniquely diversified interfaces it uses. Then again, each patch is also an opportunity to re-diversify some parts of the system.

Using diversification and obfuscation is always a race against time in the sense that the adversary will ultimately be able to guess the correct diversification key or deobfuscate the code. Dynamically changing obfuscation/diversification is a good defence, but raises complexity and performance costs of the approach.

5.2. Research gaps

As mentioned before, the main goal of conducting a systematic literature review is to collect and analyze the studies related to a field of research, in order to pinpoint the research gaps in that field.

One of the gaps we have found in this field of study is the lack of a standard metric in this field for reporting the overhead and also the effectiveness of the proposed approaches. The terms, *potency* and *resiliency*, introduced by Collberg et al. [16] discuss how much more complex the code has become in the presence of the obfuscation method; however, we believe that a standard metric to present a numeric degree is missing. Moreover, as also discussed in [276], the majority of the existing research are constructive papers and there is need for more empirical research, such as measuring the efficiency of diversification and presenting results on performance and space requirements of the obfuscation techniques. However, these concerns have not fully addressed by the research community.

Another research gap that we noted was that there are still many environments obfuscation and diversification have not been applied to yet, even though this would potentially be beneficial. In this section, we discuss these environments and present some ideas on how to utilize the two techniques as a complementary measure to the security measures these environments already have.

One of these execution environments is cloud computing and virtual environments, in general. Lately, virtual technologies have become very dominant as many enterprises and service providers are shifting towards the cloud and to deliver their services through it. Thus, we believe that due to the significance of these environments, there are needs for proactive approaches for securing their software. Obfuscation and diversification could become helpful in this manner. Our recent survey on the use of these techniques for securing cloud computing environment [293] clearly shows that there have been very limited number of studies on this domain and there is still room for more studies in this area. Also, the related works do not address the propagation issues, especially regarding propagation to higher level interfaces/APIs. One possible solution that we propose here is to diversify the internal interfaces of cloud and virtualization systems, for instance, diversifying the machine language of the virtual machines.

Therewith, container-based virtualization is drawing more attention recently, because of the advantages that it has compared to traditional hypervisor-based virtualization (such as higher efficiency in CPU, memory, and storage). We believe that this environment can also benefit from diversification techniques. However, most notably there is hardly any research related to diversification of hypervisors or containers. One potential idea is to diversify interfaces and APIs of containers, so each container would have a diverse execution environment.

IoT network is another type of environment that is becoming prevalent more and more these days. Protecting these networks is also crucial not only at network level but also at (device) application level. In the reviewed literature, obfuscation and diversification have been used very little as a security measure for this purpose. Therefore, this is another research direction to consider. We have proposed applying these two techniques on the operating systems of the IoT devices and also on the communication protocols used among them [275,322]. Then, in another study we applied diversification on Thingsee and

Raspbian operating systems [305] (the limitations of this study is discussed in Section 5.3).

These ideas could be also extended to fog computing as well. Looking at the architecture of fog computing, nodes (i.e., sensors, devices and data collectors) come at the edge of the network. Applying diversification at the nodes stops the malicious activity right at the edge before it goes up to the fog.

As mentioned earlier, diversification techniques could be applied as a complementary security approach, to the measures environments already have. We are studying applying diversification in trusted environments. More specifically, we consider diversifying the containers and storing the diversification secret inside TPM. Another idea is diversifying enclaves in a trusted execution environment such as Intel SGX.

Applying obfuscation to mobile platforms is also a topic that has received some attention recently but still requires more research. For example, Android applications, distributed in Dalvik bytecode form, are vulnerable to reverse engineering. There has been some positive development in this front, such as the ProGuard obfuscation tool being packed along with Google's Android Studio development environment and some recent publication on obfuscation in the context of mobile applications [35]. Most current studies on obfuscation in mobile environments, however, only highlight obfuscation techniques Android malware uses to evade detection. Therefore, with applications run on mobile platforms getting more prevalent, new obfuscation and diversification techniques to increase software security and research on their resilience are still needed.

To sum up, by systematically studying the two software security techniques, obfuscation and diversification, and finding the research gaps and research opportunities, we believe that both research community and industry should study and apply these techniques. There has already been some development in this direction. As we saw in Section 4.1.1, research on these techniques has drawn fair attention in software security community. This trend is also shown by several patent applications filed during the last ten years. For example, there are patents on ISR [9] and interface diversification [10]. There have also been some initiatives by Microsoft research center [119]. Still, we would like to encourage the practical application of these techniques even more – especially by industry.

5.3. Limitations of the study

As discussed earlier, a systematic literature review has a number of advantages compared to the traditional literature reviews, such as transparency, larger breadth of included studies, and reduction of research bias [4]. Nevertheless, this method has several different practical challenges and limitations as well, that we experienced in this work that made it difficult to put it into practice. First, it was very time-consuming and arduous work, due to the high number of included studies, in all the process of search, selection, data extraction, and also synthesis of the data and classification of them. Second, we selected the set of search strings manually. Thus, there is a concern that not all the materials in the field are captured. Third, the inclusion or exclusion of the studies might be biased based on the researcher's knowledge. Moreover, because of high number of authors, sometimes, each author had slightly different interpretations of the research questions. To solve the inconsistency in the screening process, we solved the disagreement cases in the meetings with the presence of all authors. Fourth, in some cases the search flow was dictated by limitations of some of the databases. As an example, in SpringerLink digital library we had to make the search in the full-text of the studies, which resulted in a huge number false positives. For other databases, we had limited the search to the title, abstract, and keywords of the studies.

In addition, like any other security measure, despite of the security benefits that the two studied techniques have, they also have some limitations such as increase in the size of the obfuscated code, and consequently increase in the execution time. In diversification method propagation of the diversification throughout the whole system from the lower levels to the upper most layers, is still a challenge and needs improvement.

Moreover, not all the diversification techniques are well suited for the tiny resource constrained devices. In other words, the diversification method should be chosen with consideration. For example, embedded systems may not always have Memory Management Units (MMU), which makes applying ASLR less worthwhile. We can perform device-specific diversification with the layout, but the embedded system requires certain offsets to be static, which means since there is no MMU we cannot hide these offsets with ASLR, which could mean all the diversification was for nothing since the exploiting code can possibly crawl through the known offsets to the offsets it needs to function. Diversification can expand the size of the system, which means the space constraints of such tiny systems come at us faster. In our previous experiments, for example, applying layout shuffling and symbol diversification required some extra space [239]. In Thingsee OS the size of binaries expanded, but not exceedingly.

6. Conclusion and future work

Obfuscation and diversification are promising software security techniques that protect computers from harmful malware. The idea of these two techniques is not to remove the security vulnerabilities, but to make it challenging for the attacker to exploit them and perform a successful attack.

In this study we reviewed the studies that were applying/studying obfuscation and diversification for improving software security. For this purpose, we systematically collected and reviewed the related studies in this field, i.e., 357 articles (See Appendix A), published between 1993 and 2017. We reported the result of study in form of analysis and classification of the captured data, and also we managed to answer the formulated research questions. We found out that these two techniques have been utilized for various aims and mitigation of various types of attacks, they can be applied at different parts of the system and at different phases of software development life-cycle. Moreover, in the literature, there exists many different techniques to obfuscate/diversify the program that each present different levels of protection, but also overhead, which depending on the need of the program could be chosen.

Moreover, by studying the existing works we pinpointed the research gaps. We concluded that the major part of the existing research works were focusing on obfuscation, and there is still room for studies on software diversification. We discussed that there exist many different environments that could still benefit from these techniques and need more focus of research, such as virtual environments, IoT, fog computing, and trusted computing. For the discussed environments, we also presented some potential ideas.

As future works, we will apply diversification on the interfaces of containers, also diversify the codes inside enclaves in trusted execution environments.

Acknowledgment

The authors gratefully acknowledge Tekes – the Finnish Funding Agency for Innovation [grant number 3772/31/2014], DIMECC Oy, and the Cyber Trust research program for their support.

Appendix A. Selected studies

The following is the list of 357 studies reviewed in this SLR, sorted based on the publication year:

1993	[11]				
1996	[12]				
1997	[13]	[14]			
1998	[15]	[16]	[17]		
1999	[18]	[19]			
2000	[20]	[21]	[22]	[23]	
2001	[24]	[25]			
2002	[26]	[27]	[28]	[29]	[30]
	[31]	[32]	[33]		
2003	[34]	[35]	[36]	[37]	[38]
	[39]	[40]	[41]	[42]	[43]
	[44]	[45]	[46]	[47]	[48]
2004	[49]	[50]	[51]	[52]	[53]
	[54]	[55]	[56]	[57]	[58]
	[59]	[60]	[61]		
2005	[62]	[63]	[64]	[65]	[66]
	[67]	[68]	[69]	[70]	
2006	[71]	[72]	[73]	[74]	[75]
	[76]	[77]	[78]	[79]	[80]
	[81]	[82]	[83]	[84]	[85]
	[86]	[87]	[88]	[89]	[90]
2007	[91]	[92]	[93]	[94]	[95]
	[96]	[97]	[98]	[99]	[100]
	[101]	[102]	[103]	[104]	[105]
	[106]	[107]	[108]		
2008	[109]	[110]	[111]	[112]	[113]
	[114]	[115]	[116]	[117]	[118]
	[119]	[120]			
2009	[121]	[122]	[123]	[124]	[125]
	[126]	[127]	[128]	[129]	[130]
	[131]	[132]	[133]	[134]	[135]
	[136]	[137]	[138]	[139]	[140]
	[141]	[142]	[143]	[144]	
2010	[145]	[146]	[147]	[148]	[149]
	[150]	[151]	[152]	[153]	[154]
	[155]	[156]	[157]	[158]	
2011	[159]	[160]	[161]	[162]	[163]
	[164]	[165]	[166]	[167]	[168]
	[169]	[170]	[171]	[172]	[173]
	[174]				
2012	[175]	[176]	[177]	[178]	[179]
	[180]	[181]	[182]	[183]	[184]
	[185]	[186]	[187]	[188]	[189]
	[190]	[191]	[192]	[193]	[194]
	[195]	[196]	[197]	[198]	[199]
	[200]				
2013	[201]	[202]	[203]	[204]	[205]
	[206]	[207]	[208]	[209]	[210]
	[211]	[212]	[213]	[214]	[215]
	[216]	[217]	[218]	[219]	[220]
	[221]	[222]			
2014	[223]	[224]	[225]	[226]	[227]
	[228]	[229]	[230]	[231]	[232]
	[233]	[234]	[235]	[236]	[237]
	[238]	[239]	[240]	[241]	[242]
	[243]	[244]	[245]	[246]	[247]
	[248]	[249]	[250]	[251]	[252]
	[253]	[254]			
2015	[255]	[256]	[257]	[258]	[259]
	[260]	[261]	[262]	[263]	[264]
	[265]	[266]	[267]	[268]	[269]
	[270]	[271]	[272]	[273]	[274]

	[275]	[276]	[277]	[278]	[279]
	[280]	[281]	[282]	[283]	[284]
	[285]	[286]	[287]	[288]	[289]
	[290]	[291]	[292]	[293]	
2016	[294]	[295]	[296]	[297]	[298]
	[299]	[300]	[301]	[302]	[303]
	[304]	[305]	[306]	[307]	[308]
	[309]	[310]	[311]	[312]	[313]
	[314]	[315]	[316]	[317]	[318]
	[319]	[320]	[321]	[322]	[323]
	[324]	[325]	[326]	[327]	[328]
	[329]	[330]	[331]	[332]	[333]
	[334]	[335]	[336]	[337]	[338]
	[339]	[340]	[341]	[342]	[343]
2017	[344]	[345]	[346]	[347]	[348]
	[349]	[350]	[351]	[352]	[353]
	[354]	[355]	[356]	[357]	[358]
	[359]	[360]	[361]	[362]	[363]
	[364]	[365]	[366]	[367]	

References

[1] C. Collberg, J. Nagra, Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection, Addison-Wesley Professional, 2009.

[2] E. Skoudis, L. Zeltser, Malware: Fighting Malicious Code, Prentice Hall Professional, Upper Saddle River, NJ 07458, 2004.

[3] B. Kitchenham, Procedures for performing systematic reviews, Technical Report TR/SE-0401, Department of Computer Science, Keele University, UK, 2004.

[4] R. Mallett, J. Hagen-Zanker, R. Slater, M. Duvendack, The benefits and challenges of using systematic reviews in international development research, J. Dev. Effectiveness 4 (3) (2012) 445–455.

[5] A. Balakrishnan, C. Schulze, Code obfuscation literature survey, Technical Report, University of Wisconsin, Madison, 2005.

[6] A. Majumdar, C. Thomborson, S. Drape, A survey of control-flow obfuscations, in: A. Bagchi, V. Atluri (Eds.), Information Systems Security, Lecture Notes in Computer Science, 4332 Springer Berlin Heidelberg, 2006, pp. 353–356.

[7] B. Kitchenham, P. Brereton, A systematic review of systematic review process research in software engineering, Inf. Softw. Technol. 55 (12) (2013) 2049–2075.

[8] IDA-PRO, (https://www.hex-rays.com/products/ida/). Accessed: 2018-07-01.

[9] J. Spradlin, Security through opcode randomization, 2012, US Patent App. 12/972,433.

[10] A. Main, M. Achim, S. Chow, H. Johnson, Y. Gu, Computer system protection by communication diversity, 2003, CA Patent App. CA 2,363,795.

[11] F.B. Cohen, Operating system protection through program evolution, Comput. Secur. 12 (6) (1993) 565–584.

[12] C. Pu, A. Black, C. Cowan, J. Walpole, C. Consel, A specialization toolkit to increase the diversity of operating systems, ICMAS Workshop Immunity-Based Systems, (1996).

[13] C. Collberg, C. Thomborson, D. Low, A taxonomy of obfuscating transformations, Technical Report 148, Department of Computer Science, The University of Auckland, New Zealand, 1997.

[14] S. Forrest, A. Somayaji, D. Ackley, Building diverse computer systems, Operating Systems, The Sixth Workshop on Hot Topics, (1997), pp. 67–72.

[15] C. Collberg, C. Thomborson, D. Low, Breaking abstractions and unstructuring data structures, Computer Languages Proceedings. International Conference on, (1998), pp. 28–38.

[16] C. Collberg, C. Thomborson, D. Low, Manufacturing cheap, resilient, and stealthy opaque constructs, in: Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '98, ACM, NY, USA, 1998, pp. 184–196.

[17] F. Hohl, Time limited blackbox security: protecting mobile agents from malicious hosts, in: G. Vigna (Ed.), Mobile Agents and Security, Lecture Notes in Computer Science, 1419 Springer Berlin Heidelberg, 1998, pp. 92–113.

[18] R.C. Linger, Systematic generation of stochastic diversity as an intrusion barrier in survivable systems software, Systems Sciences, HICSS-32. Proceedings of the 32nd Annual Hawaii International Conference on, IEEE, 1999, pp. 1–7.

[19] J. Hunt, Byte Code Protection, Java for Practitioners Practitioner Series, Springer London, 1999, pp. 427–429.

[20] V. Tam, R.K. Gupta, Using class decompilers to facilitate the security of Java applications!, Proceedings of the First International Conference on Web Information Systems Engineering. 1 (2000), pp. 153–158.

[21] C.C. Michael, A. Bartle, J. Viega, A. Hulot, N. Jarymowycz, J.R. Mills, B. Sohr, B. Arkin, Two systems for automatic software diversification, Proceedings of the DARPA Information Survivability Conference and Exposition, 2 IEEE Computer Society, 2000, p. 1220.

[22] C. Wang, J. Hill, J. Knight, J. Davidson, Software tamper resistance: obstructing static analysis of programs, Technical Report, University of Virginia, VA, USA, 2000.

[23] H. Goto, M. Mambo, K. Matsumura, H. Shizuya, Proceedings of the Information Security: Third International Workshop, ISW 2000 Wollongong, Australia, Proceedings, Springer Berlin Heidelberg, pp. 82–96.

[24] S. Chow, Y. Gu, H. Johnson, V. Zakharov, An approach to the obfuscation of control-flow of sequential computer programs, in: G. Davida, Y. Frankel (Eds.), Information Security, Lecture Notes in Computer Science, 2200 Springer Berlin Heidelberg, 2001, pp. 144–155.

[25] H. Goto, M. Mambo, H. Shizuya, Y. Watanabe, Evaluation of tamper-resistant software deviating from structured programming rules, in: V. Varadharajan, Y. Mu (Eds.), Information Security and Privacy, Lecture Notes in Computer Science, 2119 Springer Berlin Heidelberg, 2001, pp. 145–158.

[26] P. Shah, Code obfuscation for prevention of malicious reverse engineering attacks, 2002, A term paper for course ECE 578, Computer and Network Security, 12 pages.

[27] G. Wroblewski, General method of program code obfuscation, Ph.D. thesis, Wroclaw University, Poland, 2002.

[28] M. Chew, D. Song, Mitigating buffer overflows by operating system randomization, Technical Report CMU-CS-02-197, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, 2002.

[29] S. Drape, O. de Moor, G. Sittampalam, Transforming the .net intermediate language using path logic programming, in: Proceedings of the 4th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming, PPDP '02, ACM, NY, USA, 2002, pp. 133–144.

[30] L. Badger, D. Kilpatrick, B. Matt, A. Reisse, T. Van Vleck, Self-protecting mobile agents obfuscation techniques evaluation report, Technical Report 01–036, NAI Labs, 2002.

[31] T. Ogiso, Y. Sakabe, M. Soshi, A. Miyaji, Software tamper resistance based on the difficulty of interprocedural analysis, Proceedings of the Interprocedural Analysis, 3rd International Workshop on Information Security Applications (WISA), (2002), pp. 437–452.

[32] C. Bain, D. Faatz, A. Fayad, D. Williams, Diversity as a defense strategy in information systems, in: M. Gertz, E. Guldentops, L. Strous (Eds.), Integrity, Internal Control and Security in Information Systems, IFIP – The International Federation for Information Processing, 83 Springer US, 2002, pp. 77–93.

[33] C.S. Collberg, C. Thomborson, Watermarking, tamper-proofing, and obfuscation - tools for software protection, IEEE Trans. Softw. Eng. 28 (8) (2002) 735–746.

[34] C. Linn, S. Debray, Obfuscation of executable code to improve resistance to static disassembly, Proceedings of the 10th ACM Conference on Computer and Communications Security, CCS '03, NY, USA, 2003, pp. 290–299.

[35] D.C. DuVarney, V.N. Venkatakrishnan, S. Bhatkar, Self: a transparent security extension for elf binaries, Proceedings of the Workshop on New Security Paradigms, NSPW '03, ACM, USA, 2003, pp. 29–38.

[36] M. Sosonkin, G. Naumovich, N. Memon, Obfuscation of design intent in object-oriented applications, in: Proceedings of the 3rd ACM Workshop on Digital Rights Management, DRM '03, ACM, NY, USA, 2003, pp. 142–153.

[37] P. van Oorschot, Revisiting software protection, in: C. Boyd, W. Mao (Eds.), Information Security, Lecture Notes in Computer Science, 2851 Springer Berlin Heidelberg, 2003, pp. 1–13.

[38] Y. Kanzaki, A. Monden, M. Nakamura, K.-i. Matsumoto, Exploiting self-modification mechanism for program protection, Proceedings of the 27th Annual International Computer Software and Applications Conference, COMPSAC. Proceedings. (2003), pp. 170–179.

[39] E. Bhatkar, D.C. Duvarney, R. Sekar, Address obfuscation: an efficient approach to combat a broad range of memory error exploits, Proceedings of the 12th USENIX Security Symposium, (2003), pp. 105–120.

[40] L. D'Anna, B. Matt, A. Reisse, T.V. Vleck, S. Schwab, P. Leblanc, Self-protecting mobile agents obfuscation report, Technical Report 03–015, Network Associates

Laboratories, 2003.
[41] C. Collberg, G. Myles, A. Huntwork, Sandmark–a tool for software protection research, IEEE Secur. Priv. 1 (4) (2003) 40–49.
[42] G.S. Kc, A.D. Keromytis, V. Prevelakis, Countering code-injection attacks with instruction-set randomization, Proceedings of the 10th ACM Conference on Computer and Communications Security, CCS '03, NY, USA, 2003, pp. 272–280.
[43] T. Ogiso, Y. Sakabe, M. Soshi, A. Miyaji, Software obfuscation on a theoretical basis and its implementation, IEICE Trans. Fundam. Electr., Commun. Comput. Sci. 86 (1) (2003) 176–186.
[44] Y. Sakabe, M. Soshi, A. Miyaji, Java obfuscation with a theoretical basis for building secure mobile agents, in: A. Lioy, D. Mazzocchi (Eds.), Communications and Multimedia Security. Advanced Techniques for Network and Data Protection, Lecture Notes in Computer Science, 2828 Springer Berlin Heidelberg, 2003, pp. 89–103.
[45] D. Rusu, Protection methods of Java bytecode, Proceedings of the Networking in Education and Research International Conference, (2003), pp. 214–220.
[46] J. Xu, Z. Kalbarczyk, R. Iyer, Transparent runtime randomization for security, Proceedings of the 22nd International Symposium on Reliable Distributed Systems. Proceedings. (2003), pp. 260–269.
[47] A. Monden, A. Monsifrot, C. Thomborson, Obfuscated instructions for software protection, Technical Report NAIST-IS-TR2003013, Graduate School of Information Science, Nara Institute of Science and Technology, Japan, 2003.
[48] C. Dahn, S. Mancoridis, Using program transformation to secure C programs against buffer overflows, Proceedings of the 10th Working Conference on Reverse Engineering, WCRE. IEEE, 2003, pp. 323–332.
[49] S. Ring, E. Cole, Taking a lesson from stealthy rootkits, IEEE Secur. Priv. 2 (4) (2004) 38–45.
[50] H. Saputra, G. Chen, R. Brooks, N. Vijaykrishnan, M. Kandemir, M.J. Irwin, Code protection for resource-constrained embedded devices, SIGPLAN Notices 39 (7) (2004) 240–248.
[51] J.T. Chan, W. Yang, Advanced obfuscation techniques for java bytecode, J. Syst. Softw. 71 (1–2) (2004) 1–10.
[52] L. Ertaul, S. Venkatesh, JHide–a tool kit for code obfuscation, Proceedings of the 8th IASTED International Conference on Software Engineering and Applications, (2004), pp. 133–138.
[53] W. Thompson, A. Yasinsac, J.T. McDonald, Semantic encryption transformation scheme, Proceedings of the International Workshop on Security in Parallel and Distributed Systems, PDCS, (2004), pp. 516–521.
[54] D.E. Bakken, R. Parameswaran, D.M. Blough, A.A. Franz, T.J. Palmer, Data obfuscation: anonymity and desensitization of usable data sets, IEEE Secur. Priv. 2 (6) (2004) 34–41.
[55] S. Boyd, A. Keromytis, Sqlrand: preventing Sql injection attacks, in: M. Jakobsson, M. Yung, J. Zhou (Eds.), Applied Cryptography and Network Security, Lecture Notes in Computer Science, 3089 Springer Berlin Heidelberg, 2004, pp. 292–302.
[56] S. Drape, Obfuscation of abstract data types, Ph.D. thesis, University of Oxford, UK, 2004.
[57] H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N. Modadugu, D. Boneh, On the effectiveness of address-space randomization, Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS '04, NY, USA, 2004, pp. 298–307.
[58] K. Heffner, C. Collberg, The obfuscation executive, in: K. Zhang, Y. Zheng (Eds.), Information Security, Lecture Notes in Computer Science, 3225 Springer Berlin Heidelberg, 2004, pp. 428–440.
[59] D.E. Bakken, R. Parameswaran, D.M. Blough, A.A. Franz, T.J. Palmer, Data obfuscation: anonymity and desensitization of usable data sets, IEEE Secur. Priv. 2 (6) (2004) 34–41.
[60] A. Monden, A. Monsifrot, C. Thomborson, A framework for obfuscated interpretation, in: Proceedings of the 2nd Workshop on Australasian Information Security, Data Mining and Web Intelligence, and Software Internationalisation - Volume 32, ACSW Frontiers '04, Australian Computer Society, Inc., Darlinghurst, Australia, 2004, pp. 7–16.
[61] B. Anckaert, B. De Sutter, K. De Bosschere, Software piracy prevention through diversity, in: Proceedings of the 4th ACM Workshop on Digital Rights Management, DRM '04, ACM, NY, USA, 2004, pp. 63–71.
[62] J. Ge, S. Chaudhuri, A. Tyagi, Control flow based obfuscation, in: Proceedings of the 5th ACM Workshop on Digital Rights Management, DRM '05, ACM, NY, USA, 2005, pp. 83–92.
[63] L. Ertaul, S. Venkatesh, Novel obfuscation algorithms for software security, Proceedings of the International Conference on Software Engineering Research and Practice, SERP'05, (2005), pp. 209–215.
[64] C. Zhu, Z. Yin, A. Zhang, Mobile Code Security on Destination Platform, in: X. Lu, W. Zhao (Eds.), Networking and Mobile Computing, Lecture Notes in Computer Science, 3619 Springer Berlin Heidelberg, 2005, pp. 1263–1270.
[65] P.-Y. Chen, G. Kataria, R. Krishnan, Software diversity for information security. Proceedings of the Workshop on the Economics of Information Security (WEIS), Harvard University, Cambridge, MA, 2005, p. 20 pages.
[66] E.G. Barrantes, D.H. Ackley, S. Forrest, D. Stefanović, Randomized instruction set emulation, ACM Trans. Inf. Syst. Secur. 8 (1) (2005) 3–40.
[67] A. Majumdar, C. Thomborson, Securing mobile agents control flow using opaque predicates, in: R. Khosla, R. Howlett, L. Jain (Eds.), Knowledge-based intelligent information and engineering systems, Lecture Notes in Computer Science, 3683 Springer Berlin Heidelberg, 2005, pp. 1065–1071.
[68] J. Macbride, C. Mascioli, S. Marks, Y. Tang, L.M. Head, P. Ramach, A comparative study of Java obfuscators, Proceedings of the IASTED International Conference on Software Engineering and Applications (SEA), (2005), pp. 14–16.
[69] S. Bhatkar, R. Sekar, D.C. DuVarney, Efficient techniques for comprehensive protection from memory error exploits, Proceedings of the 14th Conference on USENIX Security Symposium - Vol. 14, SSYM'05, USENIX Association, Berkeley, CA, USA, 2005, p. 17.
[70] A.D. Keromytis, V. Prevelakis, A survey of randomization techniques against common mode attacks, Technical Report, Department of Computer Science, Drexel University, Philadelphia, Pennsylvania, USA, 2005.
[71] J.M. Memon, S. ul Arfeen, A. Mughal, F. Memon, Preventing reverse engineering threat in Java using byte code obfuscation techniques, Proceedings of the International Conference on Emerging Technologies. ICET '06. (2006), pp. 689–694.
[72] S. Drape, An obfuscation for binary trees, Proceedings of the TENCON. IEEE Region 10 Conference, IEEE, 2006, pp. 1–4.
[73] T. Hou, H. Chen, M. Tsai, Three control flow obfuscation methods for java software, IEEE Proc. Softw. 153 (6) (2006) 80–86.
[74] A. Majumdar, A. Monsifrot, C. Thomborson, On evaluating obfuscatory strength of alias-based transforms using static analysis, Proceedings of the International Conference on Advanced Computing and Communications, ADCOM. (2006), pp. 605–610.
[75] A. Majumdar, C. Thomborson, Manufacturing opaque predicates in distributed systems for code obfuscation, in: Proceedings of the 29th Australasian Computer Science Conference - Volume 48, ACSC '06, Australian Computer Society, Inc., Darlinghurst, Australia, 2006, pp. 187–196.
[76] Y. Kanzaki, A. Monden, M. Nakamura, K. Matsumoto, A software protection method based on instruction camouflage, Electr. Commun. Japan (Part III: Fundam. Electr. Sci.) 89 (1) (2006) 47–59.
[77] H. Yamauchi, Y. Kanzaki, A. Monden, M. Nakamura, K. Matsumoto, Software obfuscation from crackers' viewpoint. Proceedings of the ACST, (2006), pp. 286–291.
[78] M. Madou, B. Anckaert, B. De Bus, K. De Bosschere, J. Cappaert, B. Preneel, On the effectiveness of source code transformations for binary obfuscation, Proceedings of the International Conference on Software Engineering Research and Practice, SERP'06, CSREA Press, 2006, pp. 527–533.
[79] M. Madou, B. Anckaert, P. Moseley, S. Debray, B. De Sutter, K. De Bosschere, Software protection through dynamic code mutation, in: J. Song, T. Kwon, M. Yung (Eds.), Information Security Applications, Lecture Notes in Computer Science, 3786 Springer Berlin Heidelberg, 2006, pp. 194–206.
[80] B. Anckaert, M. Jakubowski, R. Venkatesan, Proteus: Virtualization for diversified tamper-resistance, Proceedings of the ACM Workshop on Digital Rights Management, DRM '06, NY, USA, 2006, pp. 47–58.
[81] B. Cox, D. E., A. Filipi, J. Rowanhill, W. Hu, J. Davidson, J. Knight, A. Nguyen-Tuong, J. Hiser, N-variant systems: a secretless framework for security through diversity. Usenix Security, 6 (2006), pp. 105–120.
[82] E. Totel, F. Majorczyk, L. Mé, Cots diversity based intrusion detection and application to web servers, in: A. Valdes, D. Zamboni (Eds.), Recent Advances in Intrusion Detection, Lecture Notes in Computer Science, 3858 Springer Berlin Heidelberg, 2006, pp. 43–62.
[83] R. Pucella, F.B. Schneider, Independence from obfuscation: a semantic framework for diversity, Proceedings of the 19th IEEE Computer Security Foundations Workshop, (2006), pp. 12 pages,–241.
[84] Y. Sakabe, M. Soshi, A. Miyaji, Java obfuscation approaches to construct tamper-resistant object-oriented programs, Inf. Media Technol. 1 (1) (2006) 134–146.
[85] W. Zhu, C. Thomborson, F.-Y. Wang, Obfuscate arrays by homomorphic functions, Proceedings of the IEEE International Conference on Granular Computing, (2006), pp. 770–773.
[86] K. Fukushima, S. Kiyomoto, T. Tanaka, An obfuscation scheme using affine transformation and its implementation, Inf. Media Technol. 1 (2) (2006) 1094–1108.
[87] M. Karnick, J. MacBride, S. McGinnis, Y. Tang, R. Ramachandran, A qualitative analysis of Java obfuscation, Proceedings of 10th IASTED International Conference on Software Engineering and Applications, TX, USA, 2006, pp. 166–171.
[88] C. Kil, J. Jim, C. Bookholt, J. Xu, P. Ning, Address space layout permutation (ASLP): Towards fine-grained randomization of commodity software, Proceedings of the 22nd Annual Computer Security Applications Conference. ACSAC '06. (2006), pp. 339–348.
[89] J. Witkowska, Biometrics, Computer Security Systems and Artificial Intelligence Applications, Springer US, Boston, MA, pp. 175–182.
[90] M. Madou, Application security through program obfuscation, Ph.D. thesis, Ghent University, Belgium, 2006.
[91] B.D. Birrer, R.A. Raines, R.O. Baldwin, B.E. Mullins, R.W. Bennington, Program fragmentation as a metamorphic software protection, Proceedings of the Third International Symposium on Information Assurance and Security, IAS. (2007), pp. 369–374.
[92] S. Drape, A. Majumdar, C. Thomborson, Slicing aided design of obfuscating transforms, Proceedings of the 6th IEEE/ACIS International Conference on Computer and Information Science, ICIS. (2007), pp. 1019–1024.
[93] N. Naeem, M. Batchelder, L. Hendren, Metrics for measuring the effectiveness of decompilers and obfuscators, Proceedings of the 15th IEEE International Conference on Program Comprehension. ICPC '07. (2007), pp. 253–258.
[94] J. Wu, X. Guo, C. Tian, H. Yin, G. Chang, The study for protecting mobile agents based on time checking technology, Proceedings of the IEEE International Conference Robotics and Biomimetics. (2007), pp. 2013–2017.
[95] B. Anckaert, M. Madou, B. De Sutter, B. De Bus, K. De Bosschere, B. Preneel, Program obfuscation: a quantitative approach, Proceedings of the 2007 ACM Workshop on Quality of Protection, QoP '07, ACM, NY, USA, 2007, pp. 15–20.
[96] A. Majumdar, S.J. Drape, C.D. Thomborson, Slicing obfuscations: design,

correctness, and evaluation, Proceedings of ACM Workshop on Digital Rights Management, DRM '07, ACM, NY, USA, 2007, pp. 70–81.
[97] S. Drape, Generalising the array split obfuscation, Inf. Sci. (Ny) 177 (1) (2007) 202–219.
[98] Y. Kinoshita, K. Kashiwagi, Y. Higami, S.-Y. Kobayashi, Development of concealing the purpose of processing for programs in a distributed computing environment, in: J. Pejaś, K. Saeed (Eds.), Advances in Information Processing and Protection, Springer US, 2007, pp. 263–269.
[99] W.F. Zhu, Concepts and techniques in software watermarking and obfuscation, Ph.D. thesis, The University of Auckland, New Zealand, 2007.
[100] I.V. Popov, S.K. Debray, G.R. Andrews, Binary obfuscation using signals, Proceedings of the 16th USENIX Security Symposium on USENIX Security Symposium, SS'07, Berkeley, CA, USA, 2007, pp. 19:1–19:16.
[101] M. Batchelder, L. Hendren, Obfuscating Java: the most pain for the least gain, in: S. Krishnamurthi, M. Odersky (Eds.), Compiler Construction, Lecture Notes in Computer Science, 4420 Springer Berlin Heidelberg, 2007, pp. 96–110.
[102] S. Praveen, P.S. Lal, Array data transformation for source code obfuscation, Proceedings of the World Academy of Science, Engineering and Technology (PWASET) Volume, 21 (2007).
[103] B. Anckaert, M. Jakubowski, R. Venkatesan, K. De Bosschere, Run-time randomization to mitigate tampering, in: A. Miyaji, H. Kikuchi, K. Rannenberg (Eds.), Advances in Information and Computer Security, Lecture Notes in Computer Science, 4752 Springer Berlin Heidelberg, 2007, pp. 153–168.
[104] S. Drape, A. Majumdar, Design and evaluation of slicing obfuscation, Technical Report, Department of Computer Science, The University of Auckland, New Zealand, 2007.
[105] X. Jiang, H.J. Wang, D. Xu, Y. Wang, RandSys: thwarting code injection attacks with system service interface randomization, Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems, SRDS. (2007), pp. 209–218.
[106] D. Bruschi, L. Cavallaro, A. Lanzi, Diversified process replicæ for defeating memory error exploits, Proceedings of the IEEE International Performance, Computing, and Communications Conference. IPCCC 2007. (2007), pp. 434–441.
[107] N. Kisserli, J. Cappaert, B. Preneel, Software security through targeted diversification, Software Security Assessments–CoBaSSA, (2007). p. 10 pages.
[108] J.C. Knight, J.W. Davidson, D. Evans, A. Nguyen-Tuong, C. Wang, Genesis: a framework for achieving software component diversity, Technical Report, University of Virginia, Charlottesville VA, USA, 2007.
[109] X. Zhang, F. He, W. Zuo, An inter-classes obfuscation method for Java program, Proceedings of the International Conference on Information Security and Assurance, 2008. ISA 2008. IEEE, 2008, pp. 360–365.
[110] J. Qin, Z. Bai, Y. Bai, Polymorphic algorithm of JavaScript code protection, Proceedings of the International Symposium on Computer Science and Computational Technology, ISCSCT '08. 1 (2008), pp. 451–454.
[111] M. Ceccato, M. Di Penta, J. Nagra, P. Falcarin, F. Ricca, M. Torchiano, P. Tonella, Towards experimental evaluation of code obfuscation techniques, in: Proceedings of the 4th ACM Workshop on Quality of Protection, QoP '08, ACM, NY, USA, 2008, pp. 39–46.
[112] P. Sivadasan, P.S. Lal, Array based Java source code obfuscation using classes with restructured arrays, arXiv:0807.4309 (2008).
[113] S. Cho, H. Chang, Y. Cho, Implementation of an obfuscation tool for C/C++ source code protection on the XScale architecture, in: U. Brinkschulte, T. Givargis, S. Russo (Eds.), Software Technologies for Embedded and Ubiquitous Systems, Springer Berlin Heidelberg, 2008, pp. 406–416.
[114] M. Jacob, M. Jakubowski, P. Naldurg, C. Saw, R. Venkatesan, The Superdiversifier: peephole individualization for software protection, in: K. Matsuura, E. Fujisaki (Eds.), Advances in Information and Computer Security, Lecture Notes in Computer Science, 5312 Springer Berlin Heidelberg, 2008, pp. 100–120.
[115] D. Dolz, G. Parra, Using exception handling to build opaque predicates in intermediate code obfuscation techniques, J. Comput. Sci. Technol. 8 (2008).
[116] S. Bhatkar, R. Sekar, Data space randomization, in: D. Zamboni (Ed.), Detection of Intrusions and Malware, and Vulnerability Assessment, Lecture Notes in Computer Science, 5137 Springer Berlin Heidelberg, 2008, pp. 1–22.
[117] H. Tamada, M. Nakamura, A. Monden, K. Matsumoto, Introducing dynamic name resolution mechanism for obfuscating system-defined names in programs, Proceedings of the International Conference on Software Engineering (IASTED SE), Innsbruck, Austria, 2008, pp. 125–130.
[118] A. Nguyen-Tuong, D. Evans, J. Knight, B. Cox, J. Davidson, Security through redundant data diversity, Proceedings of the IEEE International Conference Dependable Systems and Networks With FTCS and DCC. (2008), pp. 187–196.
[119] C. Cadar, P. Akritidis, M. Costa, J.-P. Martin, M. Castro, Data randomization, Technical Report, Microsoft Research, 2008.
[120] C. Liem, Y.X. Gu, H. Johnson, A compiler-based infrastructure for software-protection, Proceedings of the Third ACM SIGPLAN Workshop on Programming Languages and Analysis for Security, PLAS '08, ACM, NY, USA, 2008, pp. 33–44.
[121] H. Tamada, K. Fukuda, T. Yoshioka, Program incomprehensibility evaluation for obfuscation methods with queue-based mental simulation model, Proceedings of the 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel Distributed Computing (SNPD), (2012), pp. 498–503.
[122] K. Fukushima, S. Kiyomoto, T. Tanaka, Obfuscation mechanism in conjunction with tamper-proof module, Proceedings of the International Conference on Computational Science and Engineering. CSE '09. 2 (2009), pp. 665–670.
[123] S. Xiao, J. Park, Y. Ye, Tamper resistance for software defined radio software, Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference. COMPSAC '09. 1 (2009), pp. 383–391.
[124] H.-Y. Tsai, Y.-L. Huang, D. Wagner, A graph approach to quantitative analysis of control-flow obfuscating transformations, IEEE Trans. Inf. Forensics Secur. 4 (2) (2009) 257–267.
[125] Z. Tang, X. Chen, D. Fang, F. Chen, Research on Java software protection with the obfuscation in identifier renaming, Proceedings of the Fourth International Conference on Innovative Computing, Information and Control (ICICIC), (2009), pp. 1067–1071.
[126] D. Yi, A new obfuscation scheme in constructing fuzzy predicates, Proceedings of the WRI World Congress on Software Engineering. WCSE '09. 4 (2009), pp. 379–382.
[127] H. JingDe, S. Gang, Substitution encryption algorithm study for embedded mobile code protection, Proceedings of the International Conference on Communication Software and Networks, ICCSN '09. (2009), pp. 645–649.
[128] W. Jiehong, G. Fuxiang, Study of MA protection based extending inheritance hierarchy trees and time check, Proceedings of the 4th International Conference on Computer Science Education, ICCSE '09. (2009), pp. 380–384.
[129] O. Shevtsova, D. Buintsev, Methods and software for the program obfuscation, Proceedings of the International Siberian Conference on Control and Communications, SIBCON 2009. (2009), pp. 113–115.
[130] P. Sivadasan, P. SojanLal, N. Sivadasan, JDATATRANS for array obfuscation in Java source codes to defeat reverse engineering from decompiled codes, Proceedings of the 2Nd Bangalore Annual Compute Conference, COMPUTE '09, ACM, NY, USA, 2009, pp. 13:1–13:4.
[131] M. Ceccato, P. Tonella, M.D. Preda, A. Majumdar, Remote software protection by orthogonal client replacement, Proceedings of the ACM Symposium on Applied Computing, SAC '09, NY, USA, 2009, pp. 448–455.
[132] P. Wayner, Disappearing Cryptography: Information Hiding: Steganography & Watermarking, 3, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2009.
[133] Z. Lin, R. Riley, D. Xu, Polymorphing software by randomizing data structure layout, in: U. Flegel, D. Bruschi (Eds.), Detection of Intrusions and Malware, and Vulnerability Assessment, Lecture Notes in Computer Science, 5587 Springer Berlin Heidelberg, 2009, pp. 107–126.
[134] B. De Sutter, B. Anckaert, J. Geiregat, D. Chanet, K. De Bosschere, Instruction set limitation in support of software diversity, in: P. Lee, J. Cheon (Eds.), Information Security and Cryptology ICISC 2008, Lecture Notes in Computer Science, 5461 Springer Berlin Heidelberg, 2009, pp. 152–165.
[135] M. Jakubowski, C. Saw, R. Venkatesan, Tamper-tolerant software: modeling and implementation, in: T. Takagi, M. Mambo (Eds.), Advances in Information and Computer Security, Lecture Notes in Computer Science, 5824 Springer Berlin Heidelberg, 2009, pp. 125–139.
[136] J. Han, D. Gao, R.H. Deng, On the effectiveness of software diversity: a systematic study on real-world vulnerabilities, in: U. Flegel, D. Bruschi (Eds.), Detection of Intrusions and Malware, and Vulnerability Assessment, Lecture Notes in Computer Science, 5587 Springer Berlin Heidelberg, 2009, pp. 127–146.
[137] S. Drape, I. Voiculescu, The use of matrices in obfuscation, Technical Report, Oxford University Computing Laboratory, Oxford, UK, 2009.
[138] P. Sivadasan, P.S. Lal, JConstHide: A Framework for Java Source Code Constant Hiding, CoRR (2009). arXiv:0904.3458
[139] B. Anckaert, M. Jakubowski, R. Venkatesan, C.W. Saw, Runtime protection via dataflow flattening, Proceedings of the 3rd International Conference on Emerging Security Information, Systems and Technologies. SECURWARE '09. (2009), pp. 242–248.
[140] D. Williams, W. Hu, J. Davidson, J. Hiser, J. Knight, A. Nguyen-Tuong, Security through diversity: leveraging virtual machine technology, IEEE Secur. Priv. 7 (1) (2009) 26–33.
[141] H. Xu, S.J. Chapin, Address-space layout randomization using code islands, J. Comput. Secur. 17 (3) (2009) 331–362.
[142] M.H. Jakubowski, C.W. Saw, R. Venkatesan, Iterated transformations and quantitative metrics for software protection, Proceedings of the International Conference on Security and Cryptography (SECRYPT), (2009), pp. 359–368.
[143] T. László, Á. Kiss, Obfuscating C++ programs via control flow flattening, Annales Universitatis Scientarum Budapestinensis de Rolando Eötvös Nominatae, Sectio Computatorica 30 (2009) 3–19.
[144] B. Coppens, I. Verbauwhede, K.D. Bosschere, B.D. Sutter, Practical mitigations for timing-based side-channel attacks on modern x86 processors, Proceedings of the 30th IEEE Symposium on Security and Privacy, (2009), pp. 45–60.
[145] S.M. Darwish, S.K. Guirguis, M.S. Zalat, Stealthy code obfuscation technique for software security, Proceedings of the International Conference on Computer Engineering and Systems (ICCES), IEEE, 2010, pp. 93–99.
[146] T. Long, L. Liu, Y. Yu, Z. Wan, Assure high quality code using refactoring and obfuscation techniques, Proceedings of the Fifth International Conference on Frontier of Computer Science and Technology (FCST), (2010), pp. 246–252.
[147] Z. Vrba, P. Halvorsen, C. Griwodz, Program obfuscation by strong cryptography, Proceedings of the International Conference on Availability, Reliability, and Security, ARES '10, (2010), pp. 242–247.
[148] X. Guangli, C. Zheng, The code obfuscation technology based on class combination, Proceedings of the Ninth International Symposium on Distributed Computing and Applications to Business Engineering and Science (DCABES), IEEE, 2010, pp. 479–483.
[149] Q. Gu, Efficient code diversification for network reprogramming in sensor networks, in: Proceedings of the Third ACM Conference on Wireless Network Security, WiSec '10, ACM, NY, USA, 2010, pp. 145–150.
[150] M. Franz, E Unibus Pluram: massive-scale software diversity as a defense mechanism, in: Proceedings of the Workshop on New Security Paradigms, NSPW '10, ACM, NY, USA, 2010, pp. 7–16.

[151] S.S. Yau, H.G. An, Protection of users' data confidentiality in cloud computing, in: Proceedings of the Second Asia-Pacific Symposium on Internetware, Internetware '10, ACM, NY, USA, 2010, pp. 11:1–11:6.
[152] B. Lee, Y. Kim, J. Kim, binob+: a framework for potent and stealthy binary obfuscation, in: Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ASIACCS '10, NY, USA, (2010), pp. 271–281.
[153] T. Roeder, F.B. Schneider, Proactive obfuscation, ACM Trans. Comput. Syst. 28 (2) (2010) 4:1–4:54.
[154] G. Portokalidis, A.D. Keromytis, Fast and practical instruction-set randomization for commodity systems, in: Proceedings of the 26th Annual Computer Security Applications Conference, ACSAC '10, ACM, NY, USA, 2010, pp. 41–48.
[155] X. Zhang, F. He, W. Zuo, Theory and practice of program obfuscation, in: M. Crisan (Ed.), Convergence and Hybrid Information Technologies, INTECH: Croatia, 2010, pp. 277–302.
[156] Y.Y. Wei, K. Ohzeki, Obfuscation methods with controlled calculation amounts and table function, Proceedings of the International Multiconference on Computer Science and Information Technology (IMCSIT), (2010), pp. 775–780.
[157] J. Cappaert, B. Preneel, A general model for hiding control flow, in: Proceedings of the Tenth Annual ACM Workshop on Digital Rights Management, DRM '10, ACM, NY, USA, 2010, pp. 35–42.
[158] S.W. Boyd, G.S. Kc, M.E. Locasto, A.D. Keromytis, V. Prevelakis, On the general applicability of instruction-set randomization, IEEE Trans. Dependable Secure Comput. 7 (3) (2010) 255–270.
[159] F. Baiardi, D. Sgandurra, An obfuscation-based approach against injection attacks, Proceedings of the Sixth International Conference on Availability, Reliability and Security (ARES), (2011), pp. 51–58.
[160] P. Falcarin, S. Carlo, A. Cabutto, N. Garazzino, D. Barberis, Exploiting code mobility for dynamic binary obfuscation, Proceedings of the World Congress on Internet Security (WorldCIS), (2011), pp. 114–120.
[161] N. Mavrogiannopoulos, N. Kisserli, B. Preneel, A taxonomy of self-modifying code for obfuscation, Comput. Secur. 30 (8) (2011) 679–691.
[162] M. Heiderich, N. Eduardo Alberto Vela, G. Heyes, D. Lindsay, Web Application Obfuscation, Elsevier, Boston, USA, 2011.
[163] M. Christodorescu, M. Fredrikson, S. Jha, J. Giffin, End-to-end software diversification of internet services, in: S. Jajodia, A.K. Ghosh, V. Swarup, C. Wang, X.S. Wang (Eds.), Moving Target Defense, Advances in Information Security, 54 Springer NY, 2011, pp. 117–130.
[164] R. Chakraborty, S. Narasimhan, S. Bhunia, Embedded software security through key-based control flow obfuscation, in: M. Joye, D. Mukhopadhyay, M. Tunstall (Eds.), Security Aspects in Information Technology, Lecture Notes in Computer Science, 7011 Springer Berlin Heidelberg, 2011, pp. 30–44.
[165] L. Shan, S. Emmanuel, Mobile agent protection with self-modifying code, J. Signal Process. Syst. 65 (1) (2011) 105–116.
[166] A. Amarilli, S. Müller, D. Naccache, D. Page, P. Rauzy, M. Tunstall, Can code polymorphism limit information leakage? in: C. Ardagna, J. Zhou (Eds.), Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication, Lecture Notes in Computer Science, 6633 Springer Berlin Heidelberg, 2011, pp. 1–21.
[167] G. Portokalidis, A. Keromytis, Global ISR: toward a comprehensive defense against unauthorized code execution, in: S. Jajodia, A.K. Ghosh, V. Swarup, C. Wang, X.S. Wang (Eds.), Moving Target Defense, Advances in Information Security, 54 Springer NY, 2011, pp. 49–76.
[168] T. Jackson, B. Salamat, A. Homescu, K. Manivannan, G. Wagner, A. Gal, S. Brunthaler, C. Wimmer, M. Franz, Compiler-generated software diversity, in: S. Jajodia, A.K. Ghosh, V. Swarup, C. Wang, X.S. Wang (Eds.), Moving Target Defense, Advances in Information Security, 54 Springer NY, 2011, pp. 77–98.
[169] M. Ceccato, P. Tonella, Codebender: remote software protection using orthogonal replacement, IEEE Softw. 28 (2) (2011) 28–34.
[170] S. Armoogum, A. Caully, Obfuscation techniques for mobile agent code confidentiality, J. Inf. Syst. Manag. 1 (1) (2011) 83–94.
[171] P. Sivadasan, P.S. Lal, Suggesting potency measures for obfuscated arrays and usage of source code obfuscators for intellectual property protection of Java products, Proceedings of the International Conference on Information and Network Technology (ICINT), (2011).
[172] Y. Huang, A.K. Ghosh, Introducing diversity and uncertainty to create moving attack surfaces for web services, in: S. Jajodia, K.A. Ghosh, V. Swarup, C. Wang, S.X. Wang (Eds.), Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats, Springer NY, NY, 2011, pp. 131–151.
[173] D. Evans, A. Nguyen-Tuong, J. Knight, Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats, Springer NY, NY, pp. 29–48.
[174] J.C. Knight, Dependable and Historic Computing: Essays dedicated To Brian Randell on the Occasion of his 75th Birthday, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 298–312.
[175] X. Yao, J. Pang, Y. Zhang, Y. Yu, J. Lu, A method and implementation of control flow obfuscation using SEH, Proceedings of the 4th International Conference on Multimedia Information Networking and Security (MINES), (2012), pp. 336–339.
[176] A. Capiluppi, P. Falcarin, C. Boldyreff, Code defactoring: evaluating the effectiveness of Java obfuscations, Proceedings of the 19th Working Conference on Reverse Engineering (WCRE), (2012), pp. 71–80.
[177] Y. Le, H. Huo-Jiao, Research on Java bytecode parse and obfuscate tool, Proceedings of the International Conference on Computer Science Service System (CSSS), (2012), pp. 50–53.
[178] B. Rodes, Stack layout transformation: towards diversity for securing binary programs, Proceedings of the 34th International Conference on Software Engineering (ICSE), (2012), pp. 1543–1546.
[179] Z. Wang, C. Jia, M. Liu, X. Yu, Branch obfuscation using code mobility and signal, Proceedings of the IEEE 36th Annual Computer Software and Applications Conference Workshops (COMPSACW), (2012), pp. 553–558.
[180] M. Hataba, A. El-Mahdy, Cloud protection by obfuscation: techniques and metrics, Proceedings of the 2012 Seventh International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), (2012), pp. 369–372.
[181] M. Ceccato, M. Di Penta, J. Nagra, P. Falcarin, F. Ricca, M. Torchiano, P. Tonella, The effectiveness of source code obfuscation: an experimental assessment, Proceedings of the IEEE 17th International Conference on Program Comprehension, ICPC '09. (2009), pp. 178–187.
[182] A. Zambon, Aucsmith-like obfuscation of Java bytecode, Proceedings of the IEEE 12th International Working Conference on Source Code Analysis and Manipulation (SCAM), (2012), pp. 114–119.
[183] S. Qing, W. Zhi-yue, W. Wei-min, L. Jing-liang, H. Zhi-wei, Technique of source code obfuscation based on data flow and control flow transformations, Proceedings of the 7th International Conference on Computer Science Education (ICCSE), (2012), pp. 1093–1097.
[184] D. Clarke, P. Ezhilchelvan, Fortress: adding intrusion-resilience to primary-backup server systems, Proceedings of the IEEE 31st Symposium on Reliable Distributed Systems (SRDS), (2012), pp. 121–130.
[185] M. Palanques, R. Dipietro, C.d. Ojo, M. Malet, M. Marino, T. Felguera, Secure cloud browser: model and architecture to support secure web navigation, in: Proceedings of the IEEE 31st Symposium on Reliable Distributed Systems, SRDS '12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 402–403.
[186] R. Wu, P. Chen, B. Mao, L. Xie, RIM: a method to defend from JIT spraying attack, Proceedings of the Seventh International Conference on Availability, Reliability and Security (ARES), (2012), pp. 143–148.
[187] Y. Park, S.J. Stolfo, Software decoys for insider threat, in: Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, ASIACCS '12, ACM, NY, USA, 2012, pp. 93–94.
[188] R. Giacobazzi, N.D. Jones, I. Mastroeni, Obfuscation by partial evaluation of distorted interpreters, Proceedings of the ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, PEPM '12, NY, USA, 2012, pp. 63–72.
[189] C. LeDoux, M. Sharkey, B. Primeaux, C. Miles, Instruction embedding for improved obfuscation, Proceedings of the 50th Annual Southeast Regional Conference, ACM-SE '12, ACM, NY, USA, 2012, pp. 130–135.
[190] Y.-L. Huang, H.-Y. Tsai, A framework for quantitative evaluation of parallel control-flow obfuscation, Comput. Secur. 31 (8) (2012) 886–896.
[191] P. Sivadasan, P.S. Lal, Securing SQLJ source codes from business logic disclosure by data hiding obfuscation, CoRR (2012). arXiv:1205.4813
[192] R. Wartell, V. Mohan, K.W. Hamlen, Z. Lin, Binary stirring: Self-randomizing instruction addresses of legacy x86 binary code, Proceedings of the ACM Conference on Computer and Communications Security, CCS '12, NY, USA, 2012, pp. 157–168.
[193] M. Abadi, G.D. Plotkin, On protection by layout randomization, ACM Trans. Inf. Syst. Secur. 15 (2) (2012).
[194] C. Giuffrida, A. Kuijsten, A.S. Tanenbaum, Enhanced operating system security through efficient and fine-grained address space randomization, Presented as part of the 21st USENIX Security Symposium (USENIX Security 12), USENIX, Bellevue, WA, 2012, pp. 475–490.
[195] V. Pappas, M. Polychronakis, A. Keromytis, Smashing the gadgets: Hindering return-oriented programming using in-place code randomization, Proceedings of the IEEE Symposium on Security and Privacy (SP), (2012), pp. 601–615.
[196] L. Ďurfina, D. Kolář, C source code obfuscator, Kybernetika 48 (3) (2012) 494–501.
[197] J. Hiser, A. Nguyen-Tuong, M. Co, M. Hall, J.W. Davidson, Ilr: where'd my gadgets go?, Proceedings of the IEEE Symposium on Security and Privacy (SP), (2012), pp. 571–585.
[198] R. Costa, L. Pirmez, D. Boccardo, L.F. Rust, R. Machado, TinyObf: code obfuscation framework for wireless sensor networks, Proceedings of the International Conference on Wireless Networks (ICWN), The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2012, pp. 68–74.
[199] S. Rauti, V. Leppänen, Browser extension-based man-in-the-browser attacks against Ajax applications with countermeasures, Proceedings of the 13th International Conference on Computer Systems and Technologies, CompSysTech '12, ACM, NY, USA, 2012, pp. 251–258.
[200] C. Collberg, S. Martin, J. Myers, J. Nagra, Distributed application tamper detection via continuous software updates, Proceedings of the 28th Annual Computer Security Applications Conference, ACSAC '12, ACM, NY, USA, 2012, pp. 319–328.
[201] B. Bertholon, S. Varrette, S. Martinez, ShadObf: a C-source obfuscator based on multi-objective optimisation algorithms, Proceedings of the IEEE 27th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), (2013), pp. 435–444.
[202] V. Balachandran, S. Emmanuel, Potent and stealthy control flow obfuscation by stack based self-modifying code, IEEE Trans. Inf. Forensics Secur. 8 (4) (2013) 669–681.
[203] A. Homescu, S. Neisius, P. Larsen, S. Brunthaler, M. Franz, Profile-guided automated software diversity, Proceedings of the IEEE/ACM International Symposium on Code Generation and Optimization (CGO), (2013), pp. 1–11.
[204] B. Coppens, B. De Sutter, K. De Bosschere, Protecting your software updates, IEEE Secur. Priv. 11 (2) (2013) 47–54.
[205] D. Dunaev, L. Lengyel, Aspects of intermediate level obfuscation, Proceedings of the 3rd Eastern European Regional Conference on the Engineering of Computer Based Systems (ECBS-EERC), (2013), pp. 138–142.
[206] A. Gupta, M. Kirkpatrick, E. Bertino, A secure architecture design based on application isolation, code minimization and randomization, Proceedings of the IEEE

Conference on Communications and Network Security (CNS), (2013), pp. 423–429.
[207] D. Stanley, D. Xu, E. Spafford, Improved kernel security through memory layout randomization, Proceedings of the IEEE 32nd International Performance Computing and Communications Conference (IPCCC), (2013), pp. 1–10.
[208] P. Chen, R. Wu, B. Mao, JITSafe: a framework against just-in-time spraying attacks, IET Inf. Secur. 7 (9) (2013) 283–292.
[209] B. Coppens, B. De Sutter, J. Maebe, Feedback-driven binary code diversification, ACM Trans. Archit. Code Optim. 9 (4) (2013) 24:1–24:26.
[210] D.d.A.H. Marco, I. Ripoll, J.C. Ruiz, Security through emulation-based processor diversification, in: B. Akhgar, H.R. Arabnia (Eds.), Proceedings of the Emerging Trends in ICT Security, 1st edition, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2013.
[211] V. Balachandran, S. Emmanuel, Software protection with obfuscation and encryption, in: R. Deng, T. Feng (Eds.), Proceedings of the Information Security Practice and Experience, Lecture Notes in Computer Science, 7863 Springer Berlin Heidelberg, 2013, pp. 309–320.
[212] V. Samawi, A. Sulaiman, Software protection via hiding function using software obfuscation. Int. Arab J. Inf. Technol. 10 (6) (2013) 587–594.
[213] A. Kovacheva, Efficient code obfuscation for android, in: B. Papasratorn, N. Charoenkitkarn, V. Vanijja, V. Chongsuphajaisiddhi (Eds.), Advances in Information Technology, Communications in Computer and Information Science, 409 Springer International Publishing, 2013, pp. 104–119.
[214] V. Pappas, M. Polychronakis, A. Keromytis, Practical software diversification using in-place code randomization, in: S. Jajodia, A.K. Ghosh, V. Subrahmanian, V. Swarup, C. Wang, X.S. Wang (Eds.), Moving Target Defense II, Advances in Information Security, 100 Springer NY, 2013, pp. 175–202.
[215] T. Jackson, A. Homescu, S. Crane, P. Larsen, S. Brunthaler, M. Franz, Diversifying the software stack using randomized NOP insertion, in: S. Jajodia, A.K. Ghosh, V. Subrahmanian, V. Swarup, C. Wang, X.S. Wang (Eds.), Moving Target Defense II, Advances in Information Security, 100 Springer NY, 2013, pp. 151–173.
[216] A. Gupta, S. Kerr, M. Kirkpatrick, E. Bertino, Marlin: a fine grained randomization approach to defend against rop attacks, in: J. Lopez, X. Huang, R. Sandhu (Eds.), Network and System Security, Lecture Notes in Computer Science, 7873 Springer Berlin Heidelberg, 2013, pp. 293–306.
[217] M. Stewart, Algorithmic diversity for software security, CoRR (2013). arXiv:1312.3891
[218] S. Crane, P. Larsen, S. Brunthaler, M. Franz, Booby trapping software, in: Proceedings of the 2013 Workshop on New Security Paradigms Workshop, NSPW '13, ACM, NY, USA, 2013, pp. 95–106.
[219] C. Zhang, T. Wei, Z. Chen, L. Duan, L. Szekeres, S. McCamant, D. Song, W. Zou, Practical control flow integrity and randomization for binary executables, Proceedings of the IEEE Symposium on Security and Privacy (SP), (2013), pp. 559–573.
[220] M. Kanter, S. Taylor, Diversity in cloud systems through runtime and compile-time relocation, Proceedings of the IEEE International Conference on Technologies for Homeland Security (HST), (2013), pp. 396–402.
[221] L.V. Davi, A. Dmitrienko, S. Nürnberger, A.-R. Sadeghi, Gadge me if you can: Secure and efficient ad-hoc instruction-level randomization for x86 and arm, Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security, ASIA CCS '13, ACM, NY, USA, 2013, pp. 299–310.
[222] A. Homescu, S. Brunthaler, P. Larsen, M. Franz, Librando: transparent code randomization for just-in-time compilers, Proceedings of the ACM SIGSAC Conference on Computer & Communications Security, CCS '13, ACM, NY, USA, 2013, pp. 993–1004.
[223] C. Foket, B.D. Sutter, K.D. Bosschere, Pushing java type obfuscation to the limit, IEEE Trans. Dependable Secure Comput. 11 (6) (2014) 553–567.
[224] P. Larsen, S. Brunthaler, M. Franz, Security through diversity: are we there yet?, IEEE Secur. Priv. 12 (2) (2014) 28–35.
[225] S. Blazy, S. Riaud, Measuring the robustness of source program obfuscation: Studying the impact of compiler optimizations on the obfuscation of C programs, Proceedings of the 4th ACM Conference on Data and Application Security and Privacy, CODASPY '14, ACM, NY, USA, 2014, pp. 123–126.
[226] K. Lu, S. Xiong, D. Gao, Ropsteg: program steganography with return oriented programming, Proceedings of the 4th ACM Conference on Data and Application Security and Privacy, CODASPY '14, ACM, NY, USA, 2014, pp. 265–272.
[227] H.-T. Liaw, S.-C. Wei, Obfuscation for object-oriented programs: dismantling instance methods, Softw.: Pract. Exp. 44 (9) (2014) 1077–1104.
[228] M. Ceccato, M. Di Penta, P. Falcarin, F. Ricca, M. Torchiano, P. Tonella, A family of experiments to assess the effectiveness and efficiency of source code obfuscation techniques, Empir. Softw. Eng. 19 (4) (2014) 1040–1074.
[229] A. Kulkarni, R. Metta, A code obfuscation framework using code clones, in: Proceedings of the 22nd International Conference on Program Comprehension, ICPC, ACM, NY, USA, 2014, pp. 295–299.
[230] T. Tamboli, T.H. Austin, M. Stamp, Metamorphic code generation from llvm bytecode, J. Comput. Virology Hacking Tech. 10 (3) (2014) 177–187.
[231] A. Kulkarni, R. Metta, A new code obfuscation scheme for software protection, Proceedings of the IEEE 8th International Symposium on Service Oriented System Engineering (SOSE), (2014), pp. 409–414.
[232] S. Han, M. Ryu, J. Cha, B.U. Choi, Hotdol: Html obfuscation with text distribution to overlapping layers, Proceedings of the IEEE International Conference on Computer and Information Technology (CIT), (2014), pp. 399–404.
[233] S. Laurén, P. Mäki, S. Rauti, S. Hosseinzadeh, S. Hyrynsalmi, V. Leppänen, Symbol diversification of linux binaries, Proceedings of the World Congress on Internet Security (WorldCIS), IEEE, 2014, pp. 74–79.
[234] Y. Zhuang, M. Protsenko, T. Muller, F. Freiling, An(other) exercise in measuring the strength of source code obfuscation, Proceedings of the 25th International Workshop on Database and Expert Systems Applications (DEXA), (2014), pp. 313–317.
[235] L. Arockiam, S. Monikandan, Efficient cloud storage confidentiality to ensure data security, Proceedings of the International Conference on Computer Communication and Informatics (ICCCI), (2014), pp. 1–5.
[236] C. Tunc, F. Fargo, Y. Al-Nashif, S. Hariri, J. Hughes, Autonomic resilient cloud management (ARCM) design and evaluation, Proceedings of the International Conference Cloud and Autonomic Computing (ICCAC), (2014), pp. 44–49.
[237] M. Murphy, P. Larsen, S. Brunthaler, M. Franz, Software profiling options and their effects on security based diversification, in: Proceedings of the First ACM Workshop on Moving Target Defense, MTD '14, ACM, NY, USA, 2014, pp. 87–96.
[238] J. Seibert, H. Okhravi, E. Söderström, Information leaks without memory disclosures: Remote side channel attacks on diversified code, Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, CCS '14, ACM, NY, USA, 2014, pp. 54–65.
[239] R. Omar, A. El-Mahdy, E. Rohou, Arbitrary control-flow embedding into multiple threads for obfuscation: a preliminary complexity and performance analysis, in: Proceedings of the 2Nd International Workshop on Security in Cloud Computing, SCC '14, ACM, NY, USA, 2014, pp. 51–58.
[240] S. Schrittwieser, S. Katzenbeisser, Code Obfuscation Against Static and Dynamic Reverse Engineering, Springer, Berlin, Heidelberg, pp. 270–284.
[241] X. Xie, F. Liu, B. Lu, A data obfuscation based on state transition graph of mealy automata, in: D.-S. Huang, V. Bevilacqua, P. Premaratne (Eds.), Intelligent Computing Theory, Lecture Notes in Computer Science, 8588 Springer International Publishing, 2014, pp. 520–531.
[242] H. Fang, Y. Wu, S. Wang, Y. Huang, Multi-stage binary code obfuscation using improved virtual machine, in: X. Lai, J. Zhou, H. Li (Eds.), Information Security, Lecture Notes in Computer Science, 7001 Springer Berlin Heidelberg, 2011, pp. 168–181.
[243] H. Okhravi, J. Riordan, K. Carter, Quantitative evaluation of dynamic platform techniques as a defensive mechanism, in: A. Stavrou, H. Bos, G. Portokalidis (Eds.), Research in Attacks, Intrusions and Defenses, Lecture Notes in Computer Science, 8688 Springer International Publishing, 2014, pp. 405–425.
[244] C. Huang, S. Zhu, R. Erbacher, Toward software diversity in heterogeneous networked systems, in: V. Atluri, G. Pernul (Eds.), Data and Applications Security and Privacy XXVIII, Lecture Notes in Computer Science, 8566 Springer Berlin Heidelberg, 2014, pp. 114–129.
[245] V. Balachandran, N.W. Keong, S. Emmanuel, Function level control flow obfuscation for software security, Proceedings of the 8th International Conference on Complex, Intelligent and Software Intensive Systems (CISIS), (2014), pp. 133–140.
[246] C. Sahin, P. Tornquist, R. Mckenna, Z. Pearson, J. Clause, How does code obfuscation impact energy usage? Proceedings of the IEEE International Conference Software Maintenance and Evolution (ICSME), (2014), pp. 131–140.
[247] V. Balachandran, S. Emmanuel, N.W. Keong, Obfuscation by code fragmentation to evade reverse engineering, Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (SMC), (2014), pp. 463–469.
[248] P. Larsen, A. Homescu, S. Brunthaler, M. Franz, SoK: automated software diversity, Proceedings of the IEEE Symposium on Security and Privacy (SP), (2014), pp. 276–291.
[249] M. Backes, S. Nürnberger, Oxymoron: making fine-grained memory randomization practical by allowing code sharing, Proceedings of the 23rd USENIX Security Symposium (USENIX Security 14), San Diego, CA, 2014, pp. 433–447.
[250] B. Baudry, S. Allier, M. Monperrus, Tailored source code transformations to synthesize computationally diverse program variants, in: Proceedings of the International Symposium on Software Testing and Analysis, (ISSTA), ACM, NY, USA, 2014, pp. 149–159.
[251] H. Okhravi, T. Hobson, D. Bigelow, W. Streilein, Finding focus in the blur of moving-target techniques, IEEE Secur. Priv. 12 (2) (2014) 16–26.
[252] S. Rauti, J. Holvitie, V. Leppänen, Towards a diversification framework for operating system protection, in: Proceedings of the 15th International Conference on Computer Systems and Technologies, CompSysTech '14, ACM, NY, USA, 2014, pp. 286–293.
[253] S. Rauti, V. Leppänen, A proxy-like obfuscator for web application protection, Int. J. Inf. Technol. Secur. 6 (1) (2014) 39–52.
[254] S. Rauti, S. Laurén, S. Hosseinzadeh, J.-M. Mäkelä, S. Hyrynsalmi, V. Leppänen, Proceedings of the Trusted Systems: 6th International Conference, INTRUST, Beijing, China, Springer International Publishing, Cham, pp. 15–35.
[255] M. Ceccato, A. Capiluppi, P. Falcarin, C. Boldyreff, A large study on the effect of code obfuscation on the quality of java code, Empir. Softw. Eng. 20 (6) (2015) 1486–1524.
[256] A.R. Nurmukhametov, S.F. Kurmangaleev, V.V. Kaushan, S.S. Gaissaryan, Application of compiler transformations against software vulnerabilities exploitation, Progr. Comput. Softw. 41 (4) (2015) 231–236.
[257] P. Laperdrix, W. Rudametkin, B. Baudry, Mitigating browser fingerprint tracking: multi-level reconfiguration and diversification, Proceedings of the IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), (2015), pp. 98–108.
[258] K.J. Hole, Toward anti-fragility: a malware-halting technique, IEEE Secur. Priv. 13 (4) (2015) 40–46.
[259] S. Ghosh, J. Hiser, J. Davidson, Matryoshka: Strengthening software protection via nested virtual machines, Proceedings of the IEEE/ACM 1st International Workshop on Software Protection (SPRO), (2015), pp. 10–16.
[260] S. Crane, C. Liebchen, A. Homescu, L. Davi, P. Larsen, A.-R. Sadeghi, S. Brunthaler, M. Franz, Readactor: practical code randomization resilient to memory disclosure, Proceedings of the IEEE Symposium on Security and Privacy (SP), (2015), pp.

763–780.
[261] Y. Wang, J. Wei, Toward protecting control flow confidentiality in cloud-based computation, Comput. Secur. 52 (2015) 106–127.
[262] K. Hole, Diversity reduces the impact of malware, IEEE Secur. Priv. 13 (3) (2015) 48–54.
[263] L. Davi, C. Liebchen, A.-R. Sadeghi, K.Z. Snow, F. Monrose, Isomeron: code randomization resilient to (just-in-time) return-oriented programming., 2015.
[264] V. Mohan, P. Larsen, S. Brunthaler, K.W. Hamlen, M. Franz, Opaque control-flow integrity, Proceedings of the NDSS, 26 (2015), pp. 27–30.
[265] A. Höller, T. Rauter, J. Iber, C. Kreiner, Proceedings of the 7th International Workshop Software Engineering for Resilient Systems: SERENE, Proceedings, Paris, FranceSpringer International Publishing, Cham, pp. 16–30.
[266] M. Protsenko, T. Müller, Trust, Privacy and Security in Digital Business: Trustbus, Valencia, Spain, Springer International Publishing, Cham, pp. 99–110.
[267] F. Nasim, B. Aslam, W. Ahmed, T. Naeem, Proceedings of the First International Conference Codes, Cryptology, and Information Security: C2SI Morocco, Proceedings - In honor of T. Berger, Springer International Publishing, Cham, pp. 297–313.
[268] R. Fedler, S. Banescu, A. Pretschner, Proceedings of the 34th International Conference Computer Safety, Reliability, and Security: SAFECOMP, delft, The Netherlands, Proceedings, Springer International Publishing, Cham, pp. 362–371.
[269] H.P. Joshi, A. Dhanasekaran, R. Dutta, Impact of software obfuscation on susceptibility to return-oriented programming attacks, Proceedings of the 36th IEEE Sarnoff Symposium, (2015), pp. 161–166.
[270] S. Allier, O. Barais, B. Baudry, J. Bourcier, E. Daubert, F. Fleurey, M. Monperrus, H. Song, M. Tricoire, Multitier diversification in web-based software applications, IEEE Softw. 32 (1) (2015) 83–90.
[271] P. Junod, J. Rinaldini, J. Wehrli, J. Michielin, Obfuscator-LLVM: software protection for the masses, Proceedings of the 1st International Workshop on Software Protection, SPRO '15, IEEE Press, Piscataway, NJ, USA, 2015, pp. 3–9.
[272] M. Hataba, R. Elkhouly, A. El-Mahdy, Diversified remote code execution using dynamic obfuscation of conditional branches, Proceedings of the International Conference on Distributed Computing Systems Workshops (ICDCSW), (2015), pp. 120–127.
[273] B.F. Demissie, M. Ceccato, R. Tiella, Assessment of data obfuscation with residue number coding, in: Proceedings of the 1st International Workshop on Software Protection, SPRO '15, IEEE, NJ, USA, 2015, pp. 38–44.
[274] P. Larsen, S. Brunthaler, M. Franz, Automatic software diversity, IEEE Secur. Priv. 13 (2) (2015) 30–37.
[275] S. Hosseinzadeh, S. Rauti, S. Hyrynsalmi, V. Leppänen, Security in the Internet of Things through obfuscation and diversification, Proceedings of the International Conference on Computing, Communication and Security (ICCCS), (2015), pp. 1–5.
[276] A. Homescu, T. Jackson, S. Crane, S. Brunthaler, P. Larsen, M. Franz, Large-scale automated software diversity–program evolution redux, IEEE Trans. Dependable Secure Comput. 14 (2) (2015) 158–171.
[277] S. Blazy, S. Riaud, T. Sirvent, Data tainting and obfuscation: improving plausibility of incorrect taint, Proceedings of the IEEE 15th International Working Conference Source Code Analysis and Manipulation (SCAM), (2015), pp. 111–120.
[278] E. Avidan, D.G. Feitelson, From obfuscation to comprehension, in: Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension, ICPC '15, IEEE Press, Piscataway, NJ, USA, 2015, pp. 178–181.
[279] B. Abrath, B. Coppens, S. Volckaert, B.D. Sutter, Obfuscating windows DLLs, Proceedings of the IEEE/ACM 1st International Workshop on Software Protection (SPRO), (2015), pp. 24–30.
[280] M. Protsenko, S. Kreuter, T. Müller, Dynamic self-protection and tamperproofing for android apps using native code, Proceedings of the 10th International Conference Availability, Reliability and Security (ARES), (2015), pp. 129–138.
[281] Y. Kanzaki, C. Thomborson, A. Monden, C. Collberg, Pinpointing and hiding surprising fragments in an obfuscated program, in: Proceedings of the 5th Program Protection and Reverse Engineering Workshop, PPREW-5, ACM, NY, USA, 2015, pp. 8:1–8:9.
[282] S. Laurén, S. Rauti, V. Leppänen, Diversification of system calls in linux kernel, in: Proceedings of the 16th International Conference on Computer Systems and Technologies, CompSysTech '15, ACM, NY, USA, 2015, pp. 284–291.
[283] A. Jangda, M. Mishra, B. De Sutter, Adaptive just-in-time code diversification, in: Proceedings of the Second ACM Workshop on Moving Target Defense, MTD '15, ACM, NY, USA, 2015, pp. 49–53.
[284] B. Baudry, M. Monperrus, The multiple facets of software diversity: recent developments in year 2000 and beyond, ACM Comput. Surv. 48 (1) (2015) 16:1–16:26.
[285] S. Crane, A. Homescu, S. Brunthaler, P. Larsen, M. Franz, Thwarting cache side-channel attacks through dynamic software diversity, Proceedings of the NDSS, (2015), pp. 8–11.
[286] B. Baudry, S. Allier, M. Rodriguez-Cancio, M. Monperrus, Automatic software diversity in the light of test suites, CoRR (2015). arXiv:1509.00144
[287] B. Tello, M. Winterrose, G. Baah, M. Zhivich, Simulation based evaluation of a code diversification strategy, Proceedings of the 5th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH), (2015), pp. 36–43.
[288] C.K. Behera, D.L. Bhaskari, Different obfuscation techniques for code protection, Procedia Comput. Sci. 70 (2015) 757–763.
[289] S. Banescu, M. Ochoa, A. Pretschner, A framework for measuring software obfuscation resilience against automated attacks, Proceedings of the IEEE/ACM 1st International Workshop on Software Protection (SPRO), (2015), pp. 45–51.
[290] M. Hataba, A. El-Mahdy, E. Rohou, OJIT: a novel obfuscation approach using standard just-in-time compiler transformations, Proceedings of the International Workshop on Dynamic Compilation Everywhere, Netherlands, 2015.
[291] S. Banescu, A. Pretschner, D. Battré, S. Cazzulani, R. Shield, G. Thompson, Software-based protection against changeware, in: Proceedings of the 5th ACM Conference on Data and Application Security and Privacy, CODASPY '15, ACM, NY, USA, 2015, pp. 231–242.
[292] B. Baudry, S. Allier, M. Rodriguez-Cancio, M. Monperrus, DSpot: test amplification for automatic assessment of computational diversity, CoRR (2015). arXiv:1503.05807
[293] S. Hosseinzadeh, S. Hyrynsalmi, M. Conti, V. Leppänen, Security and privacy in cloud computing via obfuscation and diversification: a survey, Proceedings of the IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom), IEEE, 2015, pp. 529–535.
[294] Y. Wu, V. Suhendra, H. Saputra, Z. Zhao, Obfuscating software puzzle for denial-of-service attack mitigation, Proceedings of the IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), IEEE, 2016, pp. 115–122.
[295] A. Viticchié, L. Regano, M. Torchiano, C. Basile, M. Ceccato, P. Tonella, R. Tiella, Assessment of source code obfuscation techniques, Proceedings of the IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM), IEEE, 2016, pp. 11–20.
[296] Z. Yujia, P. Jianmin, A new compile-time obfuscation scheme for software protection, Proceedings of the International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), IEEE, 2016, pp. 1–5.
[297] K. Mahmood, D.M. Shila, Moving target defense for internet of things using context aware code partitioning and code diversification, Proceedings of the IEEE 3rd World Forum on Internet of Things (WF-IoT), IEEE, 2016, pp. 329–330.
[298] M. Styugin, V. Zolotarev, A. Prokhorov, R. Gorbil, New approach to software code diversification in interpreted languages based on the moving target technology, Proceedings of the IEEE 10th International Conference on Application of Information and Communication Technologies (AICT), IEEE, 2016, pp. 1–5.
[299] J. Gionta, W. Enck, P. Larsen, Preventing kernel code-reuse attacks through disclosure resistant code diversification, Proceedings of the IEEE Conference on Communications and Network Security (CNS), IEEE, 2016, pp. 189–197.
[300] H. Liu, Towards better program obfuscation: optimization via language models, Proceedings of the 38th International Conference on Software Engineering Companion, ICSE '16, ACM, New York, NY, USA, 2016, pp. 680–682.
[301] X. Xie, B. Lu, D. Gong, X. Luo, F. Liu, Random table and hash coding-based binary code obfuscation against stack trace analysis, IET Inf. Secur. 10 (1) (2016) 18–27.
[302] S.A. Sebastian, S. Malgaonkar, P. Shah, M. Kapoor, T. Parekhji, A study review on code obfuscation, Proceedings of the World Conference on Futuristic Trends in Research and Innovation for Social Welfare (Startup Conclave), IEEE, 2016, pp. 1–6.
[303] K. Kuang, Z. Tang, X. Gong, D. Fang, X. Chen, T. Xing, G. Ye, J. Zhang, Z. Wang, Exploiting dynamic scheduling for VM-based code obfuscation, Proceedings of the IEEE Trustcom/BigDataSE/ISPA, IEEE, 2016, pp. 489–496.
[304] H. Borck, M. Boddy, I.J.D. Silva, S. Harp, K. Hoyme, S. Johnston, A. Schwerdfeger, M. Southern, Frankencode: creating diverse programs using code clones, Proceedings of the IEEE 23rd International Conference on Software Analysis, Evolution, Reengineering (SANER), 1 IEEE, 2016, pp. 604–608.
[305] P. Mäki, S. Rauti, S. Hosseinzadeh, L. Koivunen, V. Leppänen, Interface diversification in IoT operating systems, Proceedings of the IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC), ACM, 2016, pp. 304–309.
[306] P. Wang, S. Wang, J. Ming, Y. Jiang, D. Wu, Translingual obfuscation, Proceedings of the IEEE European Symposium on Security and Privacy (EuroS P), IEEE, 2016, pp. 128–144.
[307] Y. Peng, J. Liang, Q. Li, A control flow obfuscation method for android applications, Proceedings of the 4th International Conference on Cloud Computing and Intelligence Systems (CCIS), IEEE, 2016, pp. 94–98.
[308] L. Koivunen, S. Rauti, V. Leppänen, Applying internal interface diversification to IoT operating systems, Proceedings of the International Conference on Software Security and Assurance (ICSSA), IEEE, 2016, pp. 1–5.
[309] T.Y. Chen, F.C. Kuo, W. Ma, W. Susilo, D. Towey, J. Voas, Z.Q. Zhou, Metamorphic testing for cybersecurity, Comput. (Long Beach Calif) 49 (6) (2016) 48–55.
[310] W. Liu, W. Li, Unifying the method descriptor in Java obfuscation, Proceedings of the 2nd IEEE International Conference on Computer and Communications (ICCC), IEEE, 2016, pp. 1397–1401.
[311] R. Mohsen, A.M. Pinto, Evaluating obfuscation security: a quantitative approach, in: J. Garcia-Alfaro, E. Kranakis, G. Bonfante (Eds.), Foundations and Practice of Security, Springer International Publishing, Cham, 2016, pp. 174–192.
[312] M. Ceccato, P. Falcarin, A. Cabutto, Y.W. Frezghi, C.-A. Staicu, Search based clustering for protecting software with diversified updates, in: F. Sarro, K. Deb (Eds.), Search Based Software Engineering, Springer International Publishing, Cham, 2016, pp. 159–175.
[313] D. Xu, J. Ming, D. Wu, Generalized dynamic opaque predicates: A new control flow obfuscation method, in: M. Bishop, A.C.A. Nascimento (Eds.), Information Security, Springer International Publishing, Cham, 2016, pp. 323–342.
[314] J. Aycock, Obfuscation and Optimization, Springer International Publishing, Cham, pp. 173–203.
[315] G.-l. Cai, B.-s. Wang, W. Hu, T.-z. Wang, Moving target defense: state of the art and characteristics, Front. Inf. Technol. Electr. Eng. 17 (11) (2016) 1122–1153.
[316] A. Pawlowski, M. Contag, T. Holz, Probfuscation: an obfuscation approach using probabilistic control flows, in: J. Caballero, U. Zurutuza, R.J. Rodríguez (Eds.), Detection of Intrusions and Malware, and Vulnerability Assessment, Springer International Publishing, Cham, 2016, pp. 165–185.
[317] S. Rauti, S. Laurén, J. Uitto, S. Hosseinzadeh, J. Ruohonen, S. Hyrynsalmi,

V. Leppänen, A survey on internal interfaces used by exploits and implications on interface diversification, in: B.B. Brumley, J. Röning (Eds.), Secure IT Systems, Springer International Publishing, Cham, 2016, pp. 152–168.
[318] S. Pastrana, J. Tapiador, G. Suarez-Tangil, P. Peris-López, Avrand: a software-based defense against code reuse attacks for AVR embedded devices, in: J. Caballero, U. Zurutuza, R.J. Rodríguez (Eds.), Detection of Intrusions and Malware, and Vulnerability Assessment, Springer International Publishing, Cham, 2016, pp. 58–77.
[319] R. De Keulenaer, J. Maebe, K. De Bosschere, B. De Sutter, Link-time smart card code hardening, Int. J. Inf. Secur. 15 (2) (2016) 111–130.
[320] Y. Piao, J. Jung, J.H. Yi, Server-based code obfuscation scheme for apk tamper detection, Secur. Commun. Netw. 9 (6) (2016) 457–467.
[321] V. Balachandran, Sufatrio, D.J. Tan, V.L. Thing, Control flow obfuscation for android applications, Comput. Secur. 61 (2016) 72–93.
[322] S. Hosseinzadeh, S. Hyrynsalmi, V. Leppänen, Chapter 14 - obfuscation and diversification for securing the internet of things (IoT), in: R. Buyya, A.V. Dastjerdi (Eds.), Internet of Things, Morgan Kaufmann, 2016, pp. 259–274.
[323] S. Banescu, C. Lucaci, B. Krämer, A. Pretschner, VOT4CS: A virtualization obfuscation tool for C♯, Proceedings of the ACM Workshop on Software PROtection, SPRO '16, ACM, NY, USA, 2016, pp. 39–49.
[324] S. Hosseinzadeh, S. Rauti, S. Laurén, J.-M. Mäkelä, J. Holvitie, S. Hyrynsalmi, V. Leppänen, A survey on aims and environments of diversification and obfuscation in software security, in: B. Rachev, A. Smrikarov (Eds.), Proceedings of the 17th International Conference on Computer Systems and Technologies CompSysTech'16, ACM, 2016, pp. 113–120.
[325] R. Manikyam, J.T. McDonald, W.R. Mahoney, T.R. Andel, S.H. Russ, Comparing the effectiveness of commercial obfuscators against mate attacks, in: Proceedings of the 6th Workshop on Software Security, Protection, and Reverse Engineering, SSPREW '16, ACM, NY, USA, 2016, pp. 8:1–8:11.
[326] S. Schrittwieser, S. Katzenbeisser, J. Kinder, G. Merzdovnik, E. Weippl, Protecting software through obfuscation: can it keep pace with progress in code analysis?, ACM Comput. Surv. 49 (1) (2016) 4:1–4:37.
[327] J. Coffman, D.M. Kelly, C.C. Wellons, A.S. Gearhart, Rop gadget prevalence and survival under compiler-based binary diversification schemes, in: Proceedings of the ACM Workshop on Software PROtection, SPRO '16, ACM, NY, USA, 2016, pp. 15–26.
[328] B. Abrath, B. Coppens, S. Volckaert, J. Wijnant, B. De Sutter, Tightly-coupled self-debugging software protection, in: Proceedings of the 6th Workshop on Software Security, Protection, and Reverse Engineering, SSPREW '16, ACM, NY, USA, 2016, pp. 7:1–7:10.
[329] S. Banescu, C. Collberg, V. Ganesh, Z. Newsham, A. Pretschner, Code obfuscation against symbolic execution attacks, in: Proceedings of the 32Nd Annual Conference on Computer Security Applications, ACSAC '16, ACM, NY, USA, 2016, pp. 189–200.
[330] J. Petke, Genetic improvement for code obfuscation, Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '16 Companion, ACM, NY, USA, 2016, pp. 1135–1136.
[331] M. Wu, Y. Zhang, X. Mi, Binary protection using dynamic fine-grained code hiding and obfuscation, in: Proceedings of the 4th International Conference on Information and Network Security, ICINS '16, ACM, NY, USA, 2016, pp. 1–8.
[332] H. Xu, Y. Zhou, M. Lyu, N-version obfuscation, in: Proceedings of the 2Nd ACM International Workshop on Cyber-Physical System Security, CPSS '16, ACM, NY, USA, 2016, pp. 22–33.
[333] S. Laurén, S. Rauti, V. Leppänen, An interface diversified honeypot for malware analysis, in: Proceedings of the 10th European Conference on Software Architecture Workshops, ECSAW '16, ACM, NY, USA, 2016, pp. 29:1–29:6.
[334] S. Blazy, A. Trieu, Formal verification of control-flow graph flattening, in: Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP, ACM, NY, USA, 2016, pp. 176–187.
[335] H. Koo, M. Polychronakis, Juggling the gadgets: Binary-level code randomization using instruction displacement, in: Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, ASIA CCS '16, ACM, NY, USA, 2016, pp. 23–34.
[336] K. Braden, L. Davi, C. Liebchen, A.R. Sadeghi, S. Crane, M. Franz, P. Larsen, Leakage-resilient layout randomization for mobile devices. Proceedings of the Network and Distributed System Security Symposium (NDSS), 2016 Internet Society, San Diego, CA, USA, 2016.
[337] K. Lu, W. Lee, S. Nürnberger, M. Backes, How to make ASLR win the clone wars: Runtime re-randomization. Proceedings of the NDSS, 2016 Internet Society, San Diego, CA, USA, 2016.
[338] G. Maisuradze, M. Backes, C. Rossow, What cannot be read, cannot be leveraged? revisiting assumptions of JIT-ROP defenses, Proceedings of the 25th USENIX Security Symposium, USENIX, Austin, TX, 2016, pp. 139–156.
[339] Y. Chen, Z. Wang, D. Whalley, L. Lu, Remix: on-demand live randomization, in: Proceedings of 6th ACM Conference on Data and Application Security and Privacy, CODASPY '16, ACM, NY, USA, 2016, pp. 50–61.
[340] M. Conti, S. Crane, T. Frassetto, A. Homescu, G. Koppen, P. Larsen, C. Liebchen, M. Perry, A.-R. Sadeghi, Selfrando: securing the tor browser against de-anonymization exploits, Proc. Priv. Enhancing Technol. 2016 (4) (2016) 454–469.
[341] S. Crane, A. Homescu, P. Larsen, Code randomization: haven't we solved this problem yet?, Proceedings of the IEEE Cybersecurity Development (SecDev), IEEE, 2016, pp. 124–129.
[342] D. Williams-King, G. Gobieski, K. Williams-King, J.P. Blake, X. Yuan, P. Colp, M. Zheng, V.P. Kemerlis, J. Yang, W. Aiello, Shuffler: fast and deployable continuous code re-randomization. Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16), (2016), pp. 367–382.
[343] J. Uitto, S. Rauti, V. Leppänen, Practical implications and requirements of diversifying interpreted languages, in: Proceedings of the 11th Annual Cyber and Information Security Research Conference, CISRC '16, ACM, NY, USA, 2016, pp. 14:1–14:4.
[344] N. Veeranna, B.C. Schafer, Efficient behavioral intellectual properties source code obfuscation for high-level synthesis, Proceedings of the 18th IEEE Latin American Test Symposium (LATS), IEEE, 2017, pp. 1–6.
[345] P. Kanani, K. Srivastava, J. Gandhi, D. Parekh, M. Gala, Obfuscation: maze of code, Proceedings of the 2nd International Conference on Communication Systems, Computing and IT Applications (CSCITA), IEEE, 2017, pp. 11–16.
[346] S. Wang, P. Wang, D. Wu, Composite software diversification, Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2017, pp. 284–294.
[347] H. Liu, C. Sun, Z. Su, Y. Jiang, M. Gu, J. Sun, Stochastic optimization of program obfuscation, Proceedings of the IEEE/ACM 39th International Conference on Software Engineering (ICSE), IEEE Press, Piscataway, NJ, USA, 2017, pp. 221–231.
[348] T. Cho, H. Kim, J.H. Yi, Security assessment of code obfuscation based on dynamic monitoring in android things, IEEE Access 5 (2017) 6361–6371.
[349] M. Togan, A. Feraru, A. Popescu, Virtual machine for encrypted code execution, Proceedings of the 9th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), IEEE, 2017, pp. 1–6.
[350] R. Tiella, M. Ceccato, Automatic generation of opaque constants based on the k-clique problem for resilient data obfuscation, Proceedings of the IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), IEEE, 2017, pp. 182–192.
[351] Y. Peng, G. Su, B. Tian, M. Sun, Q. Li, Control flow obfuscation based protection method for android applications, China Commun. 14 (11) (2017) 247–259.
[352] Z. Li, X.Y. Jing, X. Zhu, H. Zhang, B. Xu, S. Ying, On the multiple sources and privacy preservation issues for heterogeneous defect prediction, IEEE Trans. Softw. Eng. PP (99) (2017) 1–21.
[353] X. Chen, H. Bos, C. Giuffrida, Codearmor: virtualizing the code space to counter disclosure attacks, Proceedings of the 2017 IEEE European Symposium on Security and Privacy (EuroS P), IEEE, 2017, pp. 514–529.
[354] D. Canavese, L. Regano, C. Basile, A. Viticchié, Estimating software obfuscation potency with artificial neural networks, in: G. Livraga, C. Mitchell (Eds.), Security and Trust Management, Springer International Publishing, Cham, 2017, pp. 193–202.
[355] B. Zhao, Z. Tang, Z. Li, L. Song, X. Gong, D. Fang, F. Liu, Z. Wang, Dexpro: a bytecode level code protection system for android applications, in: S. Wen, W. Wu, A. Castiglione (Eds.), Cyberspace Safety and Security, Springer International Publishing, Cham, 2017, pp. 367–382.
[356] S. Hosseinzadeh, S. Laurén, S. Rauti, S. Hyrynsalmi, M. Conti, V. Leppänen, Obfuscation and diversification for securing cloud computing, in: V. Chang, M. Ramachandran, R.J. Walters, G. Wills (Eds.), Enterprise Security, Springer International Publishing, Cham, 2017, pp. 179–202.
[357] X. Tang, Y. Liang, X. Ma, Y. Lin, D. Gao, On the effectiveness of code-reuse-based android application obfuscation, in: S. Hong, J.H. Park (Eds.), Information Security and Cryptology – ICISC 2016, Springer International Publishing, Cham, 2017, pp. 333–349.
[358] R. Géraud, M. Koscina, P. Lenczner, D. Naccache, D. Saulpic, Generating functionally equivalent programs having non-isomorphic control-flow graphs, in: H. Lipmaa, A. Mitrokotsa, R. Matulevičius (Eds.), Secure IT Systems, Springer International Publishing, Cham, 2017, pp. 265–279.
[359] M. Morton, H. Koo, F. Li, K.Z. Snow, M. Polychronakis, F. Monrose, Defeating zombie gadgets by re-randomizing code upon disclosure, in: E. Bodden, M. Payer, E. Athanasopoulos (Eds.), Engineering Secure Software and Systems, Springer International Publishing, Cham, 2017, pp. 143–160.
[360] W. Holder, J.T. McDonald, T.R. Andel, Evaluating optimal phase ordering in obfuscation executives, in: Proceedings of the 7th Software Security, Protection, and Reverse Engineering / Software Security and Protection Workshop, SSPREW-7, ACM, NY, USA, 2017, pp. 6:1–6:12.
[361] B. Johansson, P. Lantz, M. Liljenstam, Lightweight dispatcher constructions for control flow flattening, in: Proceedings of the 7th Software Security, Protection, and Reverse Engineering / Software Security and Protection Workshop, SSPREW-7, ACM, NY, USA, 2017, pp. 2:1–2:12.
[362] K. Lim, J. Jeong, S.-j. Cho, J. Choi, M. Park, S. Han, S. Jhang, An anti-reverse engineering technique using native code and obfuscator-llvm for android applications, in: Proceedings of the International Conference on Research in Adaptive and Convergent Systems, RACS '17, ACM, NY, USA, 2017, pp. 217–221.
[363] R.N. Ismanto, M. Salman, Improving security level through obfuscation technique for source code protection using AES algorithm, in: Proceedings of 7th International Conference on Communication and Network Security, ICCNS 2017, ACM, NY, USA, 2017, pp. 18–22.
[364] M. Zhang, M. Polychronakis, R. Sekar, Protecting COTS binaries from disclosure-guided code reuse attacks, in: Proceedings of the 33rd Annual Computer Security Applications Conference, ACSAC, ACM, NY, USA, 2017, pp. 128–140.
[365] M. Pomonis, T. Petsios, A.D. Keromytis, M. Polychronakis, V.P. Kemerlis, kR˙X: Comprehensive kernel protection against just-in-time code reuse, in: Proceedings of the Twelfth European Conference on Computer Systems, EuroSys '17, ACM, NY, USA, 2017, pp. 420–436.
[366] S. Halevi, T. Halevi, V. Shoup, N. Stephens-Davidowitz, Implementing BP-obfuscation using graph-induced encoding, in: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, CCS '17, ACM, NY, USA, 2017, pp. 783–798.
[367] T. Groß, T. Müller, Protecting JavaScript apps from code analysis, in: Proceedings of the 4th Workshop on Security in Highly Connected IT Systems, SHCIS '17, ACM, NY, USA, 2017, pp. 1–6.